**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»**

**спеціальності 121 Інженерія програмного забезпечення**

**на тему: «Веб-сервіс для пошуку дистанційної роботи на основі системи рангів»**

Виконав:
студент IV курсу, групи КП-61
Пойда Андріан Васильович _____

Керівник:
Асистент кафедри ПЗКС
Юсин Яків Олексійович _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай Микола Володимирович _____

Рецензент:
В.о. завідувача кафедрою ПСТ ФІТ «КНУ
ім. Тараса Шевченка», д.т.н., с.н.с.
Порєв Геннадій Володимирович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет прикладної математики**
**Кафедра програмного забезпечення комп'ютерних систем**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

«ЗАТВЕРДЖУЮ»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2019 р.

# ЗАВДАННЯ

**на дипломний проєкт студенту**

Пойді Андріану Васильовичу

1. Тема проєкту «Веб-сервіс для пошуку дистанційної роботи на основі системи рангів», керівник проєкту Юсин Яків Олексійович, асистент, затверджені наказом по університету від «___»_____ 2020 р. №_____

2. Термін подання студентом проєкту «___» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - огляд існуючих програмних рішень;
   - обґрунтування вибору засобів реалізації;
   - опис розроблених алгоритмів та підпрограм;
   - аналіз розробленого веб-ресурсу.

5. Перелік обов'язкового графічного матеріалу:
   - діаграма прецедентів (креслення);
   - структура бази даних (креслення);
   - архітектура веб-додатка (плакат);
   - структура веб-сторінок (плакат).

6. Консультанти розділів проєкту

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | Онай М.В., доцент | | |

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1. | Вивчення літератури за тематикою проєкту | 14.11.2019 | |
| 2. | Розроблення та узгодження технічного завдання | 28.11.2019 | |
| 3. | Розроблення структури веб-ресурсу | 15.12.2019 | |
| 4. | Підготовка матеріалів першого розділу дипломного проєкту | 30.12.2019 | |
| 5. | Розроблення дизайну сторінок та графічних елементів | 03.02.2020 | |
| 6. | Підготовка матеріалів другого розділу дипломного проєкту | 20.02.2020 | |
| 7. | Програмна реалізація веб-ресурсу | 05.03.2020 | |
| 8. | Тестування веб-ресурсу | 10.04.2020 | |
| 9. | Підготовка матеріалів третього розділу дипломного проєкту | 03.05.2020 | |
| 10. | Підготовка матеріалів четвертого розділу дипломного проєкту | 10.05.2020 | |
| 11. | Підготовка графічної частини дипломного проєкту | 19.05.2020 | |
| 12. | Оформлення документації дипломного проєкту | 26.05.2020 | |

Студент Андріан ПОЙДА

Керівник проєкту Яків ЮСИН

# АНОТАЦІЯ

Даний дипломний проєкт присвячено створенню веб-додатка для пошуку дистанційної роботи на основі системи рангів. Вибір теми проєкту обґрунтований проблемами пошуку роботи для фрілансерів та ризиками втрати часу та коштів зі сторони замовників. Проєкт покликаний автоматизувати процес найму працівників та надати можливість новим користувачам (в ролі фрілансерів) сайту без проблем знайти роботу.

У роботі було проведено аналіз існуючих рішень. Визначивши переваги та недоліки аналогів, рівень їх відповідності потребам користувачів, було виявлено, що існуючі аналоги не вирішують проблеми користувачів належним чином, що вказує на доцільність створення даного веб-застосунку.

У результаті виконання аналізу поставленого завдання було проведено обґрунтування вибору СКБД та технологій для розроблення серверної та клієнтської частин веб-застосунку.

В процесі роботи над дипломним проєктом було розроблено архітектуру серверної та клієнтської частин веб-додатка, структуру бази даних.

Продукт надає можливість замовникам легко та швидко публікувати замовлення, та автоматизує процес пошуку та відбору працівників, що спрощує роботу та економить час замовника. Працівникам надається можливість перегляду всіх опублікованих замовлень, що відповідають його рангу, подавати заявку на будь-яке замовлення та відразу розпочинати роботу над проєктом, що вирішує проблему пошуку роботи для нових користувачів системи та дає можливість їм проявити себе.

# ABSTRACT

This graduation project is dedicated to creating a web application for searching a remote job based on a ranking system. The choice of topics for the project is justified by the problems of finding a job for freelancers and the risks of losing time and money for customers. The project aims to automate the hiring process and to provide an opportunity for new users (with the freelancer role) of the site to find a job without any problems.

Existing solutions analysis has been undertaken in the graduation project. Having determined the advantages and disadvantages of analogues, the level of their compliance with the needs of users, it was found that existing analogues do not solve users' problems properly, which indicates the advisability of creating this web application.

As a result of the analysis of the task, the rationale for the choice of DBMS and technologies for the development of server and client parts was carried out.

By the graduation project development, the architecture of the server and client parts of the web application and the database structure were developed.

The product provides an opportunity for customers to easily and quickly publish an order, and automates the process of searching and selecting employees which simplifies the work and saves customer’s time. Employees are given the opportunity to view all published orders corresponding to their rank, apply for any order and immediately begin work on a project. This process solves the problem of finding work for new users of the system and allows them to show themselves.

ДП.045440-01-90 Веб-сервіс для пошуку дистанційної роботи на основі системи рангів. Відомість проєкту

| Позначення | Найменування | Кіл-ть | Примітка |
|---|---|---|---|
| | Документація проєкту | | |
| | | | |
| ДП.045440-02-91 | Веб-сервіс для пошуку | 5 | |
| | дистанційної роботи на | | |
| | основі системи рангів. | | |
| | Технічне завдання | | |
| | | | |
| ДП.045440-03-81 | Веб-сервіс для пошуку | 60 | |
| | дистанційної роботи на | | |
| | основі системи рангів. | | |
| | Пояснювальна записка | | |
| | | | |
| ДП.045440-04-51 | Веб-сервіс для пошуку | 4 | |
| | дистанційної роботи на | | |
| | основі системи рангів. | | |
| | Програма та методика | | |
| | тестування | | |
| | | | |
| ДП.045440-05-34 | Веб-сервіс для пошуку | 14 | |
| | дистанційної роботи на | | |
| | основі системи рангів. | | |
| | Керівництво користувача | | |
| | | | |
| ДП.045440-06-99 | Веб-сервіс для пошуку | 1 | |
| | дистанційної роботи на | | |
| | основі системи рангів. | | |
| | Концептуальний опис | | |
| | поведінки системи. | | |
| | Діаграма прецедентів | | |
| | | | |
| | | | |
| | | | |

| Позначення | Найменування | Кіл-ть | Примітка |
|---|---|---|---|
| | | | |
| ДП.045440-07-99 | Веб-сервіс для пошуку | 1 | |
| | дистанційної роботи на | | |
| | основі системи рангів. | | |
| | Структура бази даних. | | |
| | ERD діаграма | | |
| | | | |
| ДП.045440-08-98 | Веб-сервіс для пошуку | 1 | |
| | дистанційної роботи на | | |
| | основі системи рангів. | | |
| | Компакт-диск | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

# ВЕБ-СЕРВІС ДЛЯ ПОШУКУ ДИСТАНЦІЙНОЇ РОБОТИ НА ОСНОВІ СИСТЕМИ РАНГІВ

**Технічне завдання**

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Яків ЮСИН

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Андріан ПОЙДА

2019

**ЗМІСТ**

## 1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

**Назва розробки**: Веб-сервіс для пошуку дистанційної роботи на основі системи рангів.

**Галузь застосування**: інформаційні технології.

## 2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

## 3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для вирішення проблем пошуку дистанційної роботи зі сторони працівників, та проблем пошуку та відбору працівників зі сторони замовників.

## 4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Система повинна забезпечувати такі основні функції:

- публікація замовлень користувачами з роллю замовника;
- розміщення замовлень на дошках об'явлень працівників відповідних рангів;
- подання працівником на будь-який проєкт, що відповідає його рангу;
- відправка результатів роботи замовнику;
- відправка замовником проєкту на доопрацювання працівником;
- схвалення роботи працівника та подальше завершення проєкту;

- відхилення роботи працівника та подальше повернення замовлення на дошку об’явлень;
- чат між замовником та працівником;
- перегляд та редагування профілю.

Додаткові вимоги до додатка:

- відображення для працівника інформації про ранг, кількість проєктів, що залишилось виконати для підвищення / пониження рангу;
- відображення для працівника статистики успішності виконання проєктів, активності та доходів;
- відображення для замовника списку опублікованих проєктів;
- відображення для замовника списку здійснених транзакцій.

## 5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   - «Авторизація користувача. Схема алгоритму»;
   - «Синхронізація даних. Схема роботи програмної системи».

## 6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи .....14.11.2019

Розроблення та узгодження технічного завдання .....28.11.2019

Розроблення структури web-ресурсу .....15.12.2019

Підготовка матеріалів першого розділу дипломного проєкту.....30.12.2019

Розроблення дизайну сторінок та графічних елементів.....03.02.2020

Підготовка матеріалів другого розділу дипломного проєкту.....20.02.2020

Програмна реалізація web-ресурсу.....05.03.2020

Тестування web-ресурсу .....10.04.2020

Підготовка матеріалів третього розділу дипломного проєкту.....03.05.2020

Підготовка матеріалів четвертого розділу дипломного проєкту.....10.05.2020

Підготовка матеріалів графічної частини проєкту .....19.05.2020

Оформлення технічної документації проєкту .....26.05.2020

## 7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

# ВЕБ-СЕРВІС ДЛЯ ПОШУКУ ДИСТАНЦІЙНОЇ РОБОТИ НА ОСНОВІ СИСТЕМИ РАНГІВ

**Пояснювальна записка**

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Яків ЮСИН

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Андріан ПОЙДА

2020

**ЗМІСТ**

## СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

***Автентифікація*** – перевірка достовірності пред'явленого користувачем ідентифікатора.

***Авторизація*** – надання прав на виконання дій в системі та процес перевірки прав при спробі виконання цих дій.

***Веб-браузер*** – програмне забезпечення для під'єднаного до мережі Інтернет пристрою, що дає можливість взаємодії з даними на гіпертекстовій веб-сторінці.

***Веб-фреймворк*** – каркас, призначений для створення динамічних вебдодатків та сервісів, що спрощує розробку та зменшує дублювання коду завдяки наявності готових компонентів.

***Маршрутизація URL*** – встановлення відповідності між адресою ресурсу та функцією-обробником, що викликається при запиті ресурсу.

***Шаблонізатор*** – ПЗ, що дозволяє використовувати HTML-шаблони для генерації кінцевих HTML-сторінок.

***API*** – Application Programming Interface (прикладний програмний інтерфейс).

***СУБД (СКБД)*** – система управління (керування) базами даних.

***DOM*** – Document Object Model (об’єктна модель документа).

***HTML*** – HyperText Markup Language (мова розмітки гіпертексту).

***HTTP*** – HyperText Transfer Protocol (протокол передачі гіпертексту).

***JSON*** – JavaScript Object Notation (запис об'єктів JavaScript).

***URL*** – Uniform Resource Locator (уніфікована адреса ресурсу).

***CSS*** – Cascading Style Sheets (каскадна таблиця стилів).

***SASS*** – Syntactically Awesome Style Sheets (CSS препроцесор).

***Фрілансер*** – вільнонайманець, який сам шукає собі проєкти.

# ВСТУП

На сьогоднішній день одним із способів заробітку являється фрілансинг. Згідно з останнім обзором, проведеним компанією Paychex, в США нараховується 53 мільйони фрілансерів, тобто кожен третій житель США – фрілансер. Система фрілансингу передбачає пошук віддаленої роботи через сторонні сайти. Даний спосіб являється досить зручним, оскільки для заробітку не потрібно відвідувати офіс, і все що необхідно при собі мати, це комп'ютер, та обліковий запис на одному з існуючих фрілансерських сайтів. Проте за останній час цей спосіб став дуже поширеним, на таких сайтах появилась неймовірно велика кількість користувачів, через що становиться складно знайти собі роботу, особливо для нових користувачів. В зв'язку з цим стало проблемно для клієнтів відібрати кваліфікованих працівників, які виконають необхідну роботу швидко та якісно.

Таким чином, даний дипломний проєкт присвячено вирішенню проблем існуючих фрілансерських систем, а саме:

- проблема пошуку роботи для нових працівників;
- ризики втрати часу та коштів зі сторони клієнта, у випадку, якщо працівник не є достатньо кваліфікованим;
- відсутність захисту від шахрайства як для клієнтів, так і для фрілансерів; наприклад, немає гарантії, що по завершенню проєкту клієнт виплатить кошти фрілансеру, якщо був встановлений, звісно, фіксований спосіб оплати, а не почасовий; з іншої сторони, фрілансер може вимагати почасовий спосіб оплати, після чого зробити роботу за довгий проміжок часу, та ще й не кваліфіковано, через що клієнт втратить час та гроші.

# 1. ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

## 1.1. Upwork

Upwork – найвідоміша [1] фрілансерська система. Даний сайт являється найпоширенішою платформою для пошуку роботи зі сторони фрілансерів, та пошуку працівників, зі сторони клієнтів.

Upwork має зручний інтуїтивний інтерфейс. Фрілансер, створивши обліковий запис на сайті, має можливість:

- задати загальний опис про себе та додати фотографію (рис. 1);

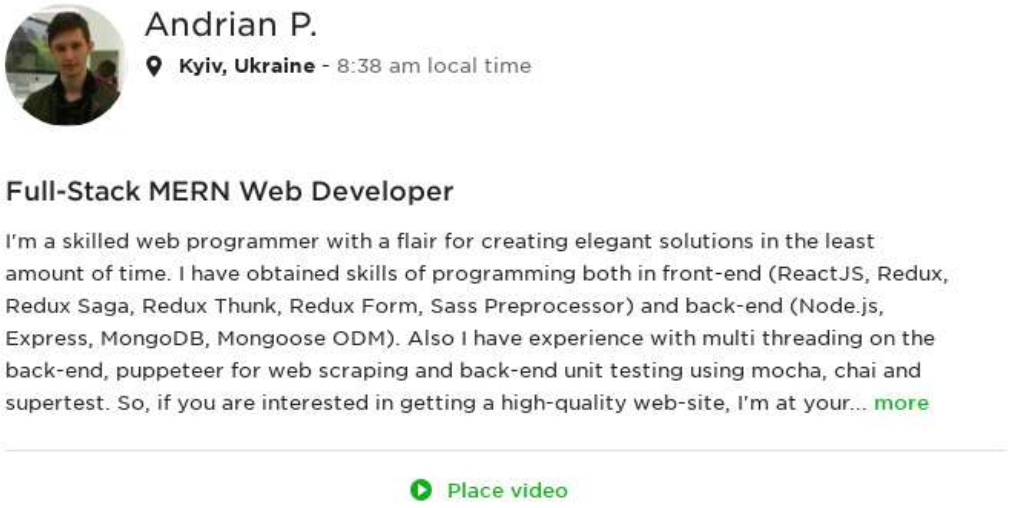



Рис. 1. Сторінка профілю

- прикріпити файли в портфоліо (резюме або виконані раніше проєкти;
- додати технології, якими він володіє;
- додати отримані сертифікати;
- додати інформацію про досвід роботи;
- додати інформацію про освіту;
- задати ціну своєї роботи.

Чим більше інформації буде додано про себе фрілансером, тим більша ймовірність того, що клієнт вибере саме його для виконання

роботи. При пошуку роботи доступні фільтри за різними параметрами (рис. 2).

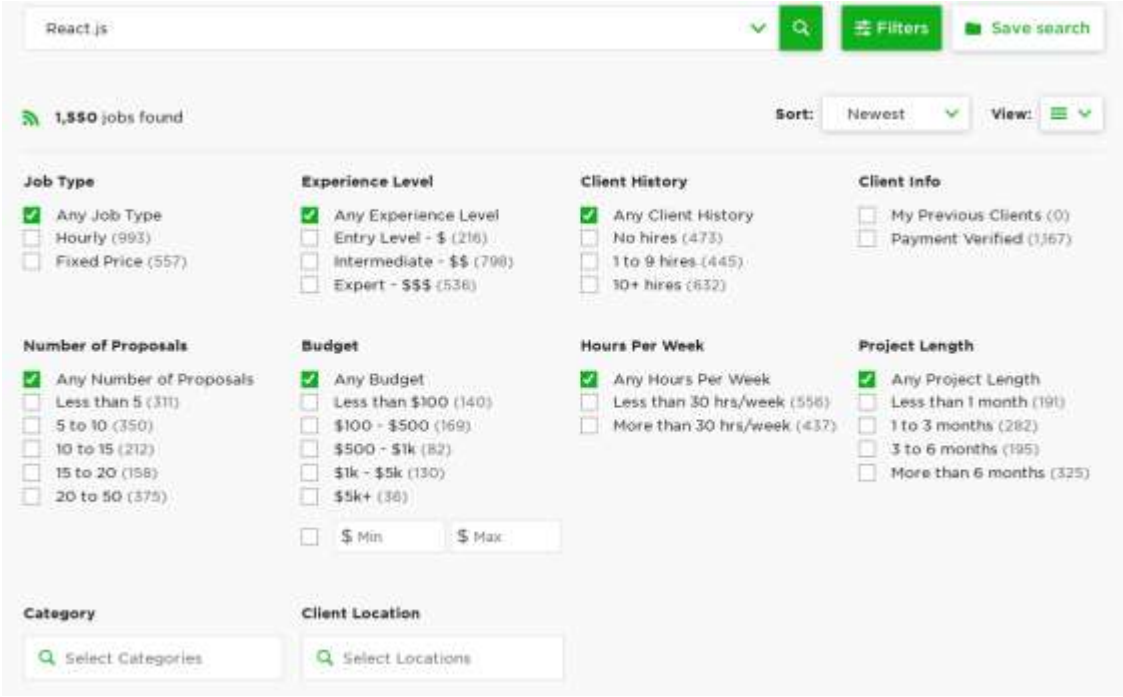



Рис. 2. Фільтри

При перегляді робіт, опублікованих клієнтами, вказується, скільки фрілансерів вже запропонували свою кандидатуру. Чим більше це число, тим менша ймовірність того, що при подачі заяви на дану роботу, клієнт хоча би розгляне профіль фрілансера.

Також на даному сайті є обмеження на подачу заяв (рис. 3).


Рис. 3. Обмеження на подачу заяв

Проте, наприклад, якщо це число рівне 60, то це не означає, що можливо подати 60 заявок. Різні вакансії мають різне значення “з’єднань”, які необхідно використати для подачі заяви. По закінченню доступних “з’єднань”, для продовження подачі заявок, необхідно буде придбати додаткові “з’єднання”.

При реєстрації в ролі клієнта, користувачу задається ряд питань, по чому формується його профіль (наприклад, сфера діяльності компанії, скільки людей працює в ній тощо). Після відповіді на питання, користувача переадресовує на сторінку публікації роботи (рис. 4).

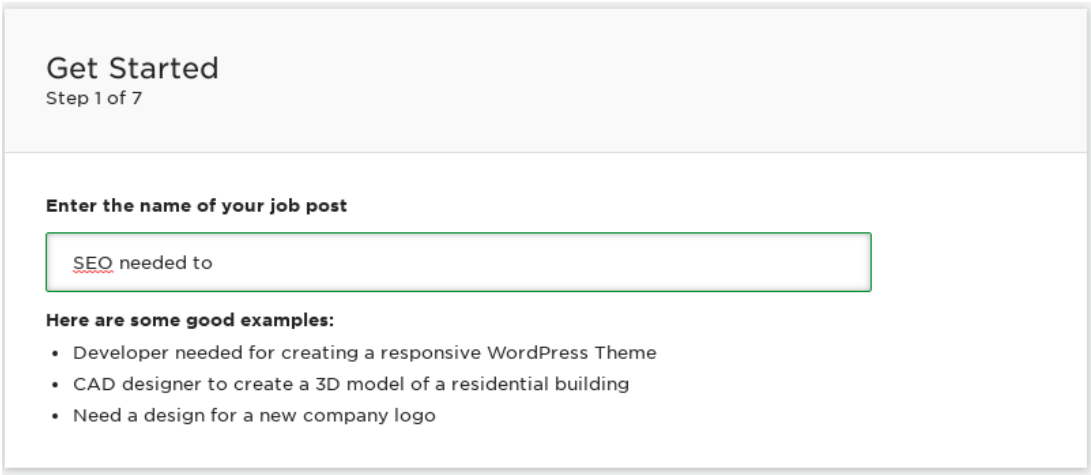



Рис. 4. Сторінка публікації роботи

При введенні назви публікації, автоматично генерується список ймовірних категорій, до яких може відноситись дана робота (рис. 5).

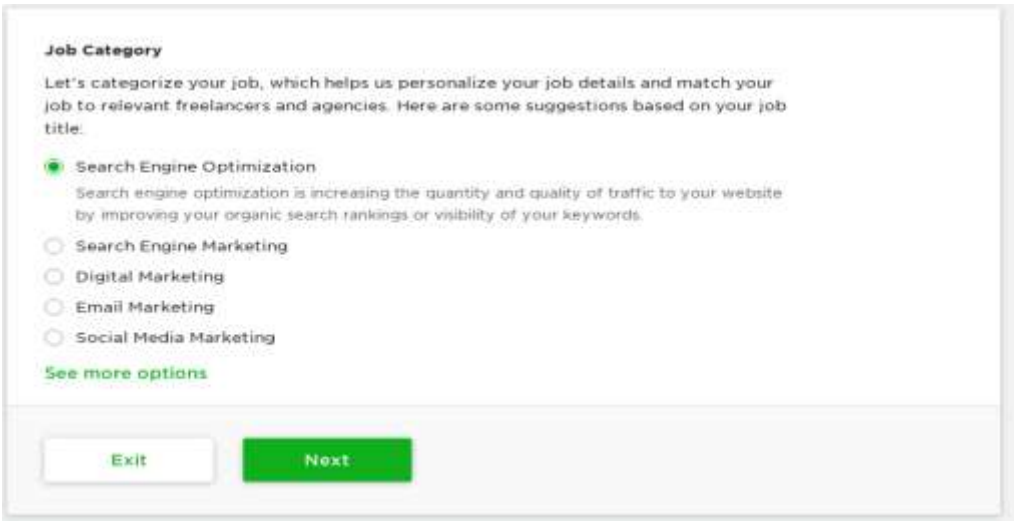



Рис. 5. Вибір категорії роботи

Для того, щоб опублікувати роботу, необхідно пройти всі етапи (рис. 6).

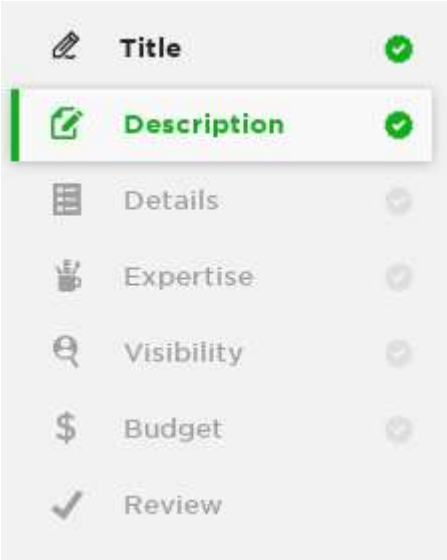



Рис. 6. Список етапів

Після цього роботу буде опубліковано, фрілансери будуть бачити її при пошуку нових робіт, або фільтруючи по навичкам чи категорії, вказаній в публікації.

Для фрілансерів є доступними рейтинг клієнта та кількість робіт, які було опубліковано на даному сайті, а також перелік певних інших параметрів. Дані параметри можуть вплинути на довіру фрілансерів до клієнта, та скласти початкове враження про нього. Проте на практиці, це не відіграє великої ролі, адже на те, чи візьметься фрілансер за дану роботу, впливає, в основному, тільки ціна та складність проєкту. Upwork надає функціональність таймера для організації роботи фрілансера. На початку роботи, фрілансер вмикає таймер, який записує час роботи та здійснює передачу коштів фрілансеру кожну годину (якщо була встановлена почасова оплата, а не фіксована), а також відстежує дії фрілансера під час роботи, робить знімки екрана, надсилає їх клієнту, та визначає відсоток активності за певний проміжок часу і будує по цьому статистику.

## 1.2. Freelance.ru

Freelance.ru – російська фрілансерська платформа. Даний сайт являється популярним на території СНД. Він, на відміну від Upwork, повністю безкоштовний, а також має певну унікальну функціональність.

На головній сторінці (навіть незареєстрованого користувача) присутня кнопка для розміщення роботи (рис. 7).

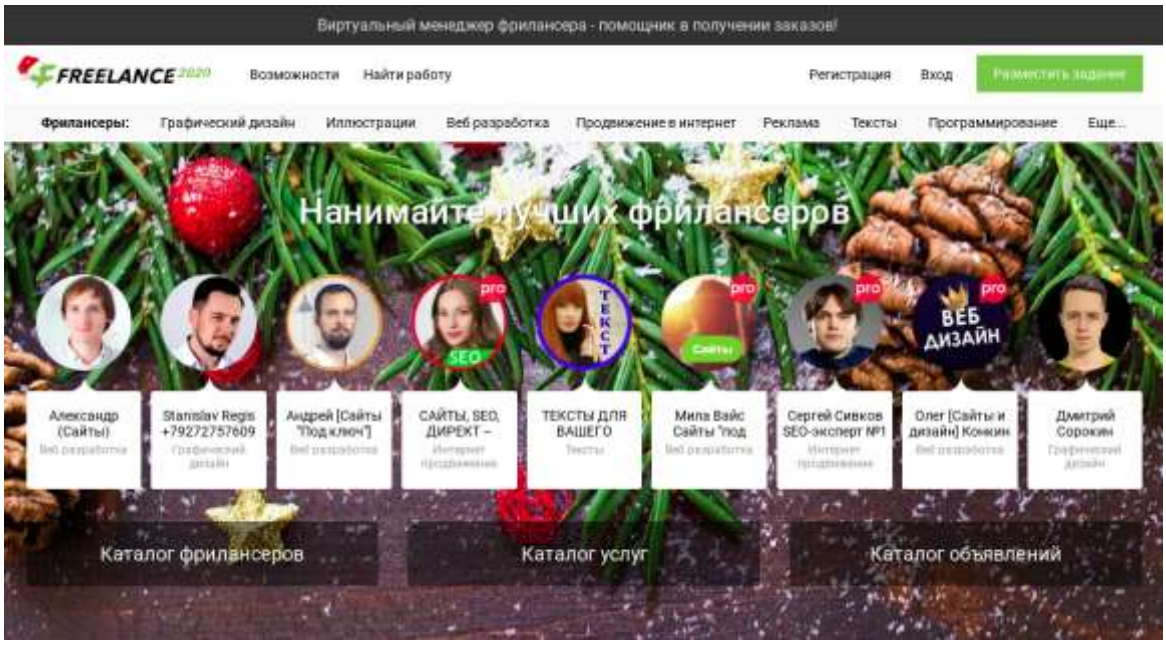



Рис. 7. Головна сторінка

На першому етапі клієнту пропонується вибрати категорію, до якої відноситься робота, а також вказати, чи ця робота дистанційна, чи необхідно працювати в офісі (рис. 8).

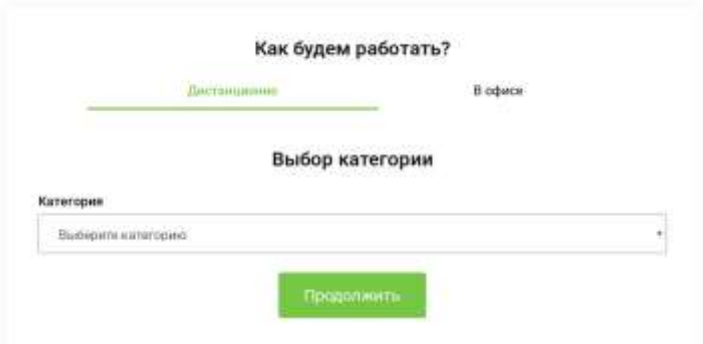



Рис. 8. Вибір категорії

При виборі категорії, випливає вікно вибору спеціалізації (рис. 9).

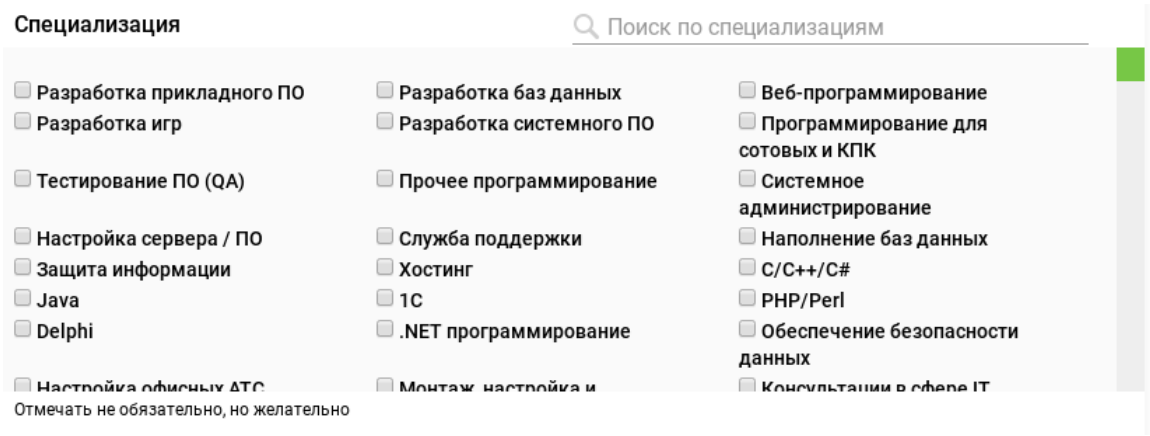



Рис. 9. Вибір спеціалізації

На другому етапі вказується назва проєкту, на третьому – опис та вимоги. Далі визначається бюджет, строки та варіант оплати (рис. 10).

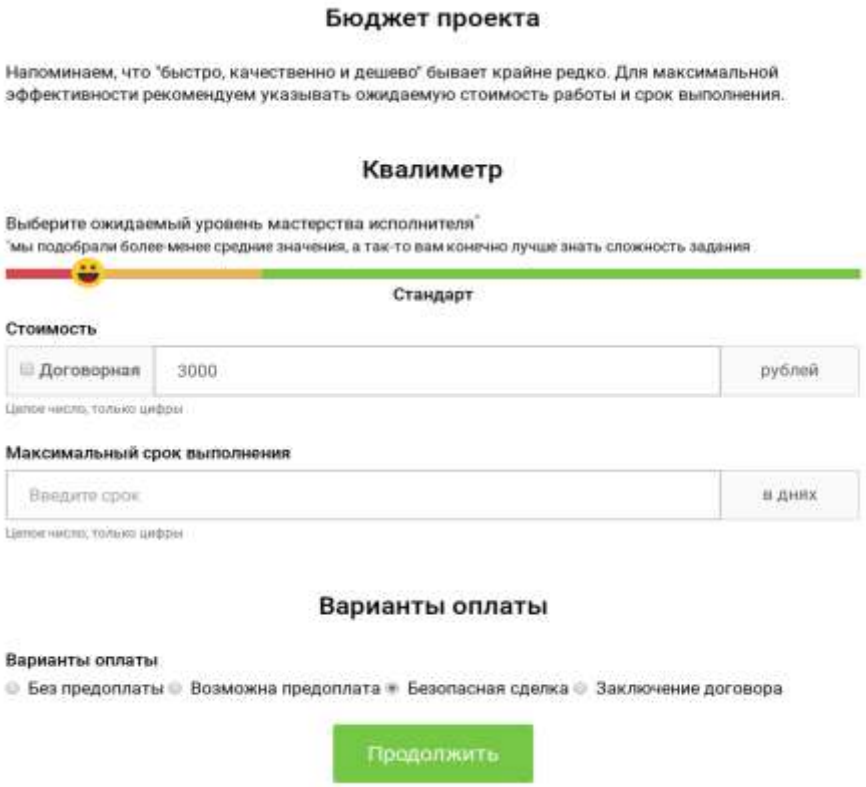



Рис. 10. Вибір бюджету, строків та варіанту оплати

На даному сайті присутні наступні унікальні функціональні можливості:

- рекомендована цінова категорія при обранні бажаного рівня кваліфікації фрілансера;
- у розділі “Варіант оплати” можна вказати “Безпечна угода”; використовуючи цю функціональність, клієнту гарантується повернення коштів в разі невиконання поставленого завдання, що забезпечує йому захист від шахрайства; з іншого боку, це також являється гарантією для фрілансера, що по завершенню проєкту клієнт виплатить йому кошти, адже вони зберігаються на резервному рахунку.

Також на сайті є функціональність під назвою “Конкурс” – клієнт може запустити конкурс по виконанню його роботи, і певна кількість спеціалістів займуться її виконанням. Завдяки цій функціональності клієнт матиме кілька варіацій виконаної роботи, та зможе вибрати найкращу (фрілансер, чию роботу вибрав клієнт, отримає весь гонорар).

Проте сайт має й певний ряд недоліків. По-перше, дизайн там є більш примітивним та незручним, в порівнянні з Upwork. Також відсутньою є локалізація, ринок проєкту розрахований тільки на СНД. Якщо говорити про функціональність резервного рахунку, то не зрозуміло, що дає гарантію фрілансеру, що клієнт не скористається функціональністю повернення коштів по завершенню проєкту. Якщо дана функціональність передбачає втручання комісії по вирішенню даної конфліктної ситуації, то цей спосіб надто не раціональний, оскільки для його реалізації необхідна велика кількість людських ресурсів, а також необхідно буде збільшувати цю кількість в разі збільшення кількості користувачів.

## 1.3. Weblancer.net

Weblancer.net – українська біржа фрілансу. Даний сайт орієнтований на Росію та Україну. Має доволі зручний інтуїтивний інтерфейс, та ряд унікальної функціональності.

На основній сторінці, незареєстрований користувач бачить поле для вводу назви проєкту та кнопку “Замовити” (рис. 11).

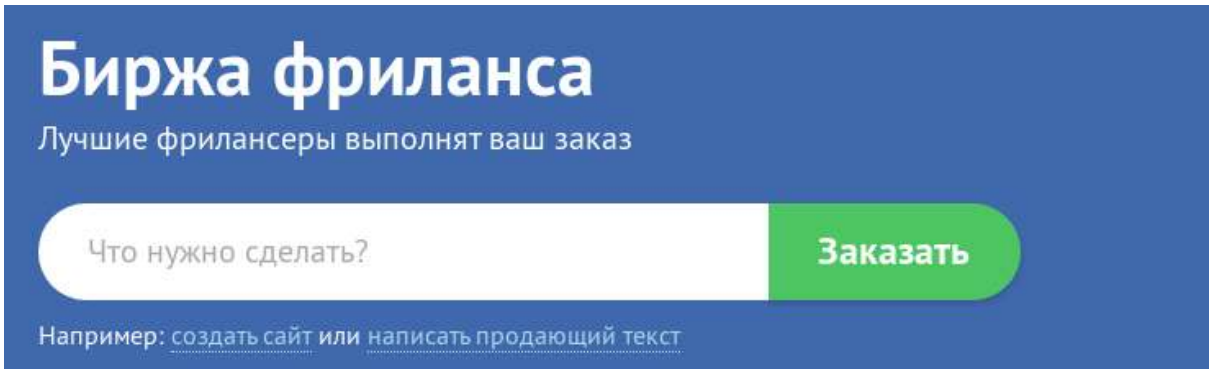



Рис. 11. Поле для вводу назви проєкту

Дана функціональність розрахована для клієнтів, та переадресовує користувача на сторінку публікації проєкту (рис. 12).

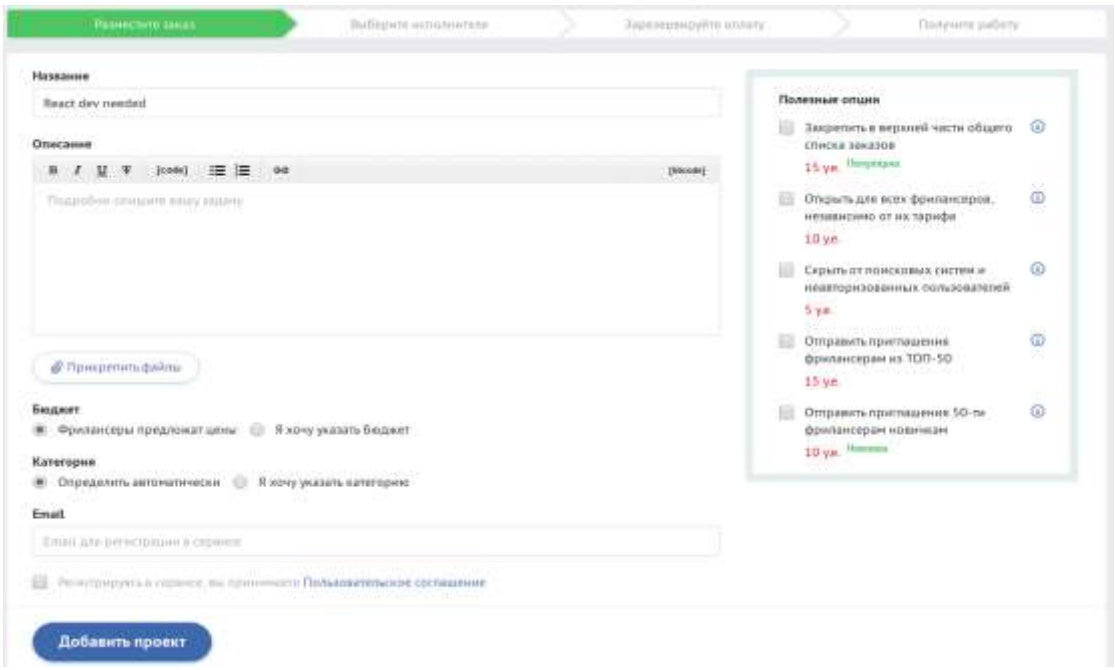



Рис. 12. Сторінка публікації проєкту

Сайт пропонує клієнтам додаткову платну функціональність, що допоможе їм швидко знайти спеціалістів. Також поле опису проєкту має зручну функціональність для форматування тексту, на відміну від Freelance.ru. Також можна прикріпити додаткові файли.

При вказівці бюджету, клієнт має можливість вказати його сам, або вибрати опцію, при якій фрілансери самі будуть пропонувати ціну за реалізацію.

Також сайт може автоматично визначити категорію проєкту, або надати можливість клієнту вказати її самостійно. Варто зазначити, що в цьому плані Upwork зручніший, так як відразу показує запропоновану категорію, а тут не зрозуміло, яка категорія вибереться для проєкту, через що дана функціональність буде використовуватись тільки тими клієнтами, для яких вибір категорії не несе ролі.

Сайт також має функціональність конкурсів. На відміну від Freelance.ru, тут передбачається певна кількість призових місць, за які надається винагорода.

Серед недоліків можна також виділити локалізацію (система орієнтована на країни СНД) та відсутність інтегрованої платіжної системи Visa/Mastercard/Payoneer. Для переведення коштів необхідно користуватись такими системами, як Webmoney або Яндекс.Деньги.

## 1.4. Freelancer.com

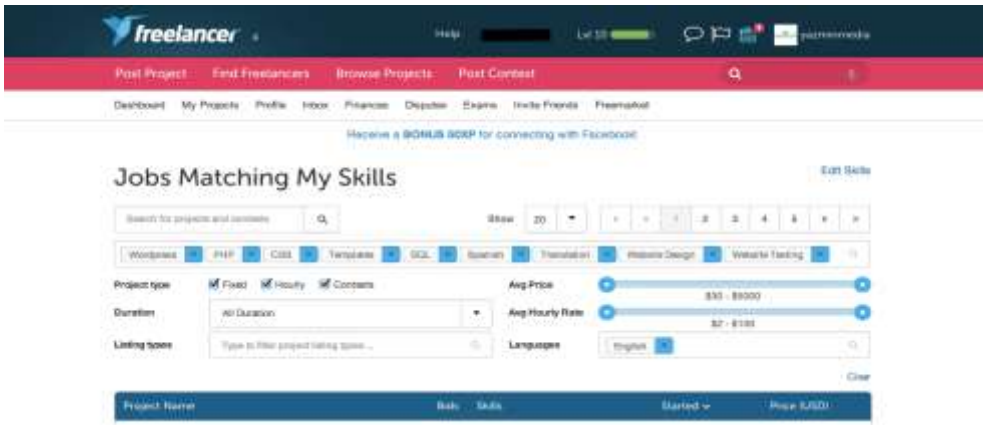



Рис. 13. Головна сторінка

Freelancer.com – міжнародна біржа фрілансу. Даний сайт являється міжнародною біржою фрілансу, тому розрахований на всі країни, також, на відміну від Upwork, на сайті присутній російськомовний та україномовний інтерфейс. Причиною цього являється те, що не малий відсоток користувачів сайту є жителями країн СНД.

Також ця платформа має функціональні можливості соціальної мережі: є можливість надсилати запрошення своїм друзям із мережі на приєднання до проєкту. Також на сайті присутні новини, блоги, можна купляти статус, вибирати задачі з бонус-балами та інші оригінальні функціональні можливості, проте більша частина корисної функціональності є платною.

Те, скільки буде роботи у фрілансера, напряму залежить від рейтингу, а решта базової функціональності тут така ж, як і на інших сайтах. Тому, хоча дана біржа і має достатньо оригінальний набір можливостей, основні проблеми фрілансерських систем залишаються не вирішеними.

## 1.5. Висновки

На фрілансерських сайтах, розглянутих вище, діє приблизно така ж схема, як і при реальному пошуку роботи – клієнтам необхідні працівники з досвідом, і в даному випадку розглядається досвід конкретно на даному сайті. Тобто, клієнт зазвичай вибирає фрілансерів з високим рейтингом та позитивними відгуками від попередніх клієнтів. Тому для нових користувачів сайту найкращим варіантом буде, якщо клієнт знайде його сам та запропонує роботу. Це можливо, адже клієнти також беруть до уваги резюме та портфоліо (прикріплені проєкти), проте діло в тому, що при пошуку фрілансерів, в основному, беруться до уваги працівники тільки на перших сторінках, а там знаходяться працівники з найвищим рейтингом та відгуками. Тому ймовірність того, що клієнт сам запропонує роботу новому користувачу-фрілансеру, доволі мала.

Окрім проблем пошуку роботи для фрілансерів, існують також ризики для клієнтів. Можна найняти фрілансера з найвищими рейтингами, і зазвичай, він виконає роботу максимально швидко та якісно. Проте проблема в тому, що не всі клієнти мають достатньо грошей для найму таких працівників, а також не гарантовано, що даний фрілансер виконає роботу конкретно так, як хотів клієнт. А якщо приймати на роботу працівників з меншим досвідом та кількістю відгуків, невідомо, наскільки якісно даний фрілансер буде виконувати роботу. Тому в існуючих системах для клієнтів є завжди ризик втрати часу та грошей.

На основі переваг та недоліків розглянутих бірж для фрілансингу, сформовано ряд вимог до системи, що розробляється:

- повинна бути імплементована система оплати (наприклад Payoneer API);
- сайт повинен містити систему рангів, як вирішення проблеми пошуку роботи для нових робітників; зареєструючись на сайті, фрілансер матиме найнижчий ранг, а після успішного виконання певної кількості проєктів, ранг буде підвищуватись, та навпаки – при певній кількості провалених проєктів, ранг знижуватиметься; даний підхід забезпечить розділення фрілансерів на категорії по рівню кваліфікації, що спростить клієнту процес пошуку працівника: все, що необхідно, це вибрати відповідний ранг, на основі власного бюджету; система рангів також спростить пошук роботи для фрілансерів, оскільки кожен користувач матиме змогу вибрати для себе будь-яку роботу на дошці об’явлень;
- система повинна мати резервний рахунок, на якому зберігатимуться кошти клієнта, при поданні фрілансером на виставлену клієнтом роботу, та переводитись на рахунок фрілансера при несхваленні клієнтом виконаної роботи;

- в разі несхвалення клієнтом виконаної роботи, проєкт повинен повернутись на дошку оголошень; система також додасть даний проєкт в число не успішно виконаних проєктів фрілансером, та при досягненні цього числа ліміту, понизить його ранг.

# 2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

## 2.1. Мови програмування

### *2.1.1. JavaScript*

JavaScript [2] – прототипна мова програмування, код якої інтерпретується та компілюється під час виконання. Вона має декілька парадигм та підтримує об'єктно-орієнтований, імперативний та декларативний (тобто функціональне програмування) стилі. Стандартом для JavaScript є ECMAScript. Насамперед відома як скриптова мова для веб-сторінок – надає можливість керувати браузером, динамічно оновлювати DOM-дерево та асинхронно обмінюватись даними із сервером. Проте мова також використовується для програмування на стороні серверу, розробки стаціонарних та мобільних застосунків, тощо.

Для розробки клієнтської частини (код, що виконується на пристрої кінцевого користувача) прийнято використовувати технології для розробки односторінкових застосунків, такі як React, Vue або Angular.

React – найпопулярніша [3] бібліотека для розробки односторінкових застосунків, розроблена Facebook в 2013 році.

Дана бібліотека має наступні переваги:

- є легкою для розуміння та вивчення завдяки простому дизайну, детальній документації та використанню JSX [4] (зручний та лаконічний HTML-подібний синтаксис для визначення деревовидних структур);
- є швидкою, завдяки реалізації React Virtual DOM дерева, яке надає можливість взаємодіяти не з DOM деревом напряму, а з його копією; Virtual DOM – об’єктне представлення HTML-документа, і при потребі оновити дані на сторінці, взаємодія здійснюється саме з ним; після оновлення Virtual DOM здійснюється його порівняння з DOM, після чого оновлюються необхідна частина сторінки; такий спосіб оптимізує процес

оновлення даних, а також надає можливість оновлювати лише необхідні частини сторінки, замість повного її оновлення;

- надає можливість зберігання даних в глобальному об’єкті – Redux Store; redux – бібліотека, яка реалізовує Flux архітектуру; всі необхідні дані зберігаються в глобальному об’єкті – «store»; до цих даних можна доступитись з будь-якого компоненту завдяки селекторам; для оновлення даних викликаються відповідні методи (так звані «actions» обробники), що передають дані в редуктори (reducers), які в залежності від переданих в них «action» та «state» оновлюють сховище даних;
- надає можливість винесення всіх асинхронних запитів поза компоненти, завдяки Redux Saga/Thunk; дані бібліотеки дають можливість зробити код більш зрозумілим та «чистим» завдяки винесенню всієї логіки взаємодії з сервером поза React компоненти;
- React реалізовує концепції функціонального програмування, створюючи простий для тестування та багаторазово використовуваний код.

Проте React також має наступні недоліки:

- змішування шаблонів з логікою (JSX) може збити з толку деяких розробників, особливо тих, які раніше користувались Angular;
- React, на відміну від готових фреймворків, таких як Angular, при ініціалізації проєкту не містить необхідних бібліотек / функціональних можливостей, їх необхідно встановлювати додатково, що може бути складно для розробників, які раніше користувались фреймворками;
- деякі фреймворки для UI дизайну дають обмежені можливості налаштування стилів під власні потреби, та вимагають

використання «inline styles», що призводить до погіршення читабельності коду.

Angular – JavaScript-фреймворк, що реалізує шаблон MVC та призначений для розробки односторінкових додатків.

Переваги:

- використовується разом з TypeScript, що розширює можливості при написанні об’єктно-орієнтованого коду;
- структура та архітектура, що створені для великої масштабованості проєкту;
- одностороння прив’язка даних, яка забезпечує виняткову поведінку додатка, що зводить до мінімуму ризик виникнення можливих помилок;
- детальна документація, яка дає можливість розробнику отримати всю необхідну інформацію;
- MVVM (Model-View-ViewModel), яка дозволяє розробникам працювати над одним і тим же розділом програми, використовуючи один і той же набір даних.

Недоліки:

- різновидність різних структур (Injectables, Components, Pipes, Modules тощо) ускладнює процес вивчення в зрівнянні з React чи Vue;
- відносно низька продуктивність.

Vue – JavaScript-фреймворк, який більше всього пасує для створення високо адаптованих користувацьких інтерфейсів та складних односторінкових додатків.

Переваги:

- детальна та зрозуміла документація;
- адаптивність – може бути здійснений швидкий перехід від інших фреймворків (React/Angular) через схожість з ними з точки зору дизайну та архітектури;

- інтеграція – можна використовувати як для односторінкових додатків, так і для більш складних веб-інтерфейсів додатків; невеликі інтерактивні елементи можна без проблем інтегрувати в існуючу інфраструктуру без негативних наслідків;
- масштабованість – надає можливість легко розробляти великі шаблони багатократного використання, які можуть бути реалізовані майже за той же час, що і більш прості;
- невеликий розмір – близько 20 КБ, що дозволяє досягнути кращої продуктивності в зрівнянні з іншими платформами.

Недоліки:

- нехватка ресурсів, так як Vue займає невелику долю на ринку в зрівнянні з React чи Angular;
- ризик виникнення невирішуваних проблем інтеграції у великі проєкти.

JavaScript також використовується для реалізації серверної частини проєктів. Для цього існує Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків. Запускає V8 JavaScript Engine, ядро Google Chrome, поза межами браузера, що робить її високопродуктивною.

Node.js [5] запускається в одному потоці (проте є також можливість використовувати декілька ядер за допомогою «Child processes API»), на відміну від більш поширеної нині моделі паралелізму, в якій використовуються потоки ОС. Мережі на основі потоків доволі не ефективні та складні у використанні. Більше того, Node.js розробники не хвилюються про блокування процесів, так як блокувань в Node.js нема. Практично жодна функція не виконує напряму ввід/вивід, тому процес ніколи не блокується. Оскільки нічого не блокується, масштабовані процеси раціонально розробляти на Node.js.

Node.js по дизайну схожа на такі системи, як Ruby's Event Machine та Python's Twisted. Node.js просуває модель подій дещо дальше. В інших

системах завжди є блокуючий виклик для запуску циклу обробки подій. Зазвичай поведінка визначається через так звані «callback» функції на початку скрипту, а в кінці сервер запускається через блокуючий виклик. В Node.js нема такого виклику, натомість відбувається вхід в цикл обробки подій після виконання сценарію вводу. Node.js виходить з циклу обробки подій, коли не залишається «callback» функцій для виконання. Така поведінка схожа на JavaScript в браузері – цикл подій скритий від користувача.

### *2.1.2. Java*

Java – об'єктно-орієнтована мова програмування загального призначення, спроектована таким чином, щоб мати якомога менше залежностей реалізації. У web-програмуванні Java використовується як для розробки серверної частини, так і для «full-stack» розробки. Для цього використовуються Spring Projects [6] – набір технологій, які включають все необхідне для веб-розробки:

- Spring Framework надає можливість писати код з використанням Dependency Injection, а також технології для веб-розробки, тестування, інтеграції тощо; для веб-розробки в основному використовується Spring Boot, який побудований на основі Spring Framework, вбудованих HTTP серверів (Tomcat, Jetty) та XML конфігурацій, що дає можливість легко розгорнути проєкт та розпочати розробку;
- Spring Data надає можливість взаємодіяти з реляційною чи нереляційною базою даних, або хмарними сервісами;
- Spring Security надає можливість реалізації функціональності авторизації/автентифікації;
- JavaServer Pages (JSP) – серверні Java EE компоненти, що генерують відповіді на запити від клієнтів; фактично JSP являється шаблонізатором, тобто вбудованим Java кодом в

HTML сторінку; для виділення Java коду використовуються наступні знаки: <% та %>; JSP компілюється в Java сервлет, після чого згенерований сервлет генерує відповідь на запит клієнта; для збереження ідентифікатора сесії використовуються cookies, local storage або URL.

### *2.1.3. Python*

Python [7] – динамічна, інтерпретована, високорівнева мова програмування, орієнтована на підвищення продуктивності роботи програмістів та читабельності коду. Python підтримує об'єктно-орієнтоване, структурне, функціональне, імперативне та аспектно-орієнтоване (парадигма програмування, побудована на ідеї розділення функціональності з метою подальшого розбиття програми на модулі) програмування. Основними напрямами використання мови являється машинне навчання або робота з «big data». Проте мова також використовується для розробки веб-застосунків, та містить набір технологій, які надають можливість та спрощують процес веб-розробки за допомогою Python.

Одним із Python фреймворків для веб-розробки являється Django – веб-орієнтований фреймворк, який надає таку функціональність, як: авторизація, панель управління, структура для завантаження файлів, шаблонізатор та маршрутизацій URL-адрес. Всі ці функціональні можливості стають доступні відразу після встановлення та налаштування фреймворку, що позбавляє програміста рутинної роботи, та дає можливість перейти відразу до розробки головної функціональності. Хоча даний фреймворк і не являється гнучким, проте методи, які він надає, є безпечними та протестованими.

Іншим популярним фреймворком для веб-розробки на Python являється Flask. Всі компоненти цього модуля призначені для серверної розробки. Flask являється мікро-фреймворком, що робить його легковагим,

що оптимізує ресурси веб-сервера та розширює можливості програміста, проте заставляє його працювати на менш абстрактному рівні, та замислюватись над реалізацією функціональності, яка в Django прописана за замовчуванням.

Pyramid – ще один популярний Python фрейморк. Він надає більше можливостей, ніж Flask, але далеко не весь спектр можливостей, що надає Django. Однак Pyramid являється одним з найбільш гнучких фреймворків, оскільки він не заставляє програміста використовувати конкретний підхід до вирішення задачі. Модуль надає підтримку для реалізації автентифікації та маршрутизації URL-адрес, проте підключатись до бази даних або сховища необхідно із застосуванням сторонніх бібліотек.

## 2.2. Бази даних

### *2.2.1. PostgreSQL*

PostgreSQL – потужна [8] об’єктно-реляційна система керування базами даних (СКБД), яка використовує та розширює мову SQL в комбінації з іншими функціями, які безпечно зберігають та масштабують складні робочі навантаження даних.

PostgreSQL [9] зберігає дані в таблицях та використовує динамічні та статичні схеми для реляційних даних та сховища. PostgreSQL управляє своїм паралелізмом, дотримуючись концепції MVVC. PostgreSQL надає багато можливостей, таких як реплікація, індексація, схеми, широкий спектр типів даних, онлайн резервування даних, процедурна мова тощо.

PostgreSQL використовує FDW, що дозволяє створювати зовнішні таблиці в базі даних PostgreSQL, які посилаються на дані з інших джерел. При посиланні запиту до зовнішньої таблиці, FDW буде робити запит на отримання даних із зовнішнього джерела та повертати результати так, якби дані знаходились всередині бази даних PostgreSQL.

Для взаємодії з PostgreSQL та іншими об’єктно-реляційними системами через Javascript, використовується Sequelize ORM.

Переваги [10]:

- дана система керування базами даних являється масштабованою та може опрацьовувати терабайти даних;
- підтримує JSON;
- доступний ряд інтерфейсів.

Недоліки:

- нестабільна, складна документація;
- падіння швидкості при масових операціях та запитах на читання.

### *2.2.2. DynamoDB*

DynamoDB [11] – NoSQL система керування базами даних, надається Amazon як частина Amazon Web Services. DynamoDB дозволяє своїм користувачам створювати бази даних, що здатні зберігати та повертати великі об’єми даних. Управління трафіком відбувається автоматично за рахунок розподілення даних та динамічного управління трафіком запитів кожного клієнта, це також допомагає підтримувати та підвищувати продуктивність. Система в основному відома своїми низькими затримками та масштабованістю.

Переваги та особливості:

- масштабована система, що дозволяє зберігати дані на сховищі Amazon;
- DynamoDB використовує таблиці, основними компонентами являються елементи та атрибути; в DynamoDB таблиця складається з набору елементів, кожен з яких містить набір атрибутів; первинні ключі використовуються для унікальної ідентифікації кожного елементу в таблиці, а вторинні – для забезпечення більшої гнучкості при посиланні запитів;
- підтримує обмежену кількість мов програмування: Java, JavaScript, Swift, Node.js, .NET, PHP та Python;

- високий рівень безпеки даних – для доступу надання до системи використовується IAM (Identity Access Management) – веб-сервіс, що дозволяє безпечно контролювати доступ до AWS ресурсів через пару «access/secret key».

### *2.2.3. MongoDB*

MongoDB [12] – одна з найбільш популярних документо-орієнтованих систем керування базами даних, яка використовується для зберігання великих об'ємів даних. Замість використання таблиць, як в традиційних реляційних СКБД, дані в MongoDB організовані у вигляді JSON документів та колекцій. Колекції містять набір документів, що є еквівалентом таблиць в реляційних СКБД. Документи складаються з пар «ключ-значення», що є базовою одиницею представлення даних в MongoDB.

Дані в MongoDB зберігаються у форматі BSON (binary-serialized JSON), що розширює JSON реалізацію, добавляючи додаткові типи даних (наприклад, дата). Система також забезпечує валідацію даних. MongoDB не вимагає використання схем, тобто кожен документ може мати унікальний набір полів в одній колекції.

Найпопулярніші ORM: mongoose (для Node.js) та mongoid (для Ruby on Rails).

Особливості MongoDB [13]:

- зберігання даних у форматі BSON;
- швидка обробка CRUD операцій;
- індексація – можливо індексувати будь-яке поле в документі;
- реплікація – підтримка «Master Slave» реплікації;
- дублювання даних – MongoDB може запускатись на кількох серверах, дані при цьому дублюються для підтримки працездатності системи та безперебійної роботи в разі виникнення помилок (наприклад, відмова обладнання);

- автоматичне балансування навантаження завдяки даним, розміщеним в сегментах;
- підтримка MapReduce та агрегації.

## 2.3. Висновки

У даному розділі розглянуто різні технології та мови програмування, які використовуються при розробці web-застосунків. Отримана інформація систематизована з метою використання в подальшій роботі для визначення конкретних інструментів, що будуть використовуватись при розробці web-сервісу для пошуку дистанційної роботи на основі системи рангів. В якості мови програмування обрано Javascript, Node.js для розроблення сервера, та React для розроблення клієнтської частини, оскільки:

- для розроблення клієнтської частини з максимально високою продуктивністю доцільніше всього використовувати сучасний JavaScript фреймворк, такий як React, оскільки дана бібліотека використовує віртуальне DOM-дерево, що дозволяє розробляти динамічний та оптимізований користувацький інтерфейс;
- для розроблення невеликих проєктів React пасує більше за Angular, а мажорна доля на ринку дає явну перевагу над таким непопулярним фреймворком, як Vue, в плані «community»;
- процес освоєння та розроблення, використовуючи вищевказані технології, доволі швидкий в зрівнянні з рештою технологій;
- проста і зрозуміла документація.

Серед баз даних було вибрано MongoDB та Mongoose ODM для взаємодії з нею через Javascript, оскільки:

- швидкість – висока продуктивність при виконанні простих запитів;
- гнучкість – в MongoDB можна додавати поля або колонки без шкоди для існуючих даних, їх структури і продуктивності СКБД;

- нескладна структура бази даних з невеликою кількістю зв’язків, через що використання нереляційної бази даних являється доцільнішим;
- MongoDB має всю необхідну для реалізації проєкту функціональність та детальну документацію, а також являється найпоширенішою документо-орієнтованою системою керування баз даних, а отже велике «community».

# 3. ОПИС РОЗРОБЛЕНИХ АЛГОРИТМІВ ТА ПІДПРОГРАМ

## 3.1. Загальний опис системи

Розроблена система є веб-платформою для пошуку дистанційної роботи, яка побудована на основі системи рангів.

Основною ціллю розроблення даного проєкту являлось вирішення існуючих проблем платформ для пошуку дистанційної роботи. Як розглянуто в першому розділі, існуючі платформи дають можливість фрілансерам подавати заявки на вакансії замовників, проте процес найму залишається за замовником. Тобто замовнику необхідно буде якимось чином визначити кваліфікацію та відібрати найкращого кандидата. Звісно, судити він буде в першу чергу по рейтингу, з чого випливає одна з проблем в існуючих фрілансерських платформах – складність пошуку роботи для нових користувачів сайту. Окрім цього, вибираючи за рейтингом, клієнту приходиться платити більше за роботу, яку може виконати і новачок. Також високий рейтинг не завжди означає високу кваліфікацію працівника, тобто звідси випливає наступна проблема існуючих платформ – ризик втрати часу та грошей зі сторони клієнта. Даний проєкт орієнтований на вирішення цих проблем шляхом автоматизації процесу найму працівників. Платформа буде аналогією компаній з надання аутсорсингових послуг – єдине, що потребується від замовника, це визначитись з бюджетом до проєкту, а відповідальність за підбір працівника система бере на себе.

### *3.1.1. Функціональність для замовника*

Зареєструвавшись на сайті в ролі замовника, в користувача з’являється можливість відразу опублікувати замовлення. Для цього йому потрібно відкрити відповідне вікно, ввести необхідні дані, та оплатити роботу (кошти зберігатимуться на резервному фонді до моменту підтвердження замовником роботи працівника). Після успішного

здійснення оплати, замовлення з’являється на дошці об’явлень фрілансерів відповідних рангів (ранг визначається в залежності від бюджету, вказаного замовником), а статус замовлення становиться «pending».

Після того, як працівник, бажаючий виконати роботу, подасть на неї заявку, статус замовлення зміниться на «active», а між клієнтом та замовником з’явиться чат, через який вони зможуть підтримувати зв’язок з метою вияснення деталей проєкту.

Працівник буде зобов’язаний завершити роботу до кінцевого строку, вказаного замовником, та відправити йому результати. Після отримання результатів роботи працівника, в замовника появляються наступні можливості:

- схвалити роботу, після цього статус замовлення зміниться на «closed», працівнику переведуться кошти, а проєкт зарахується як успішно завершений;
- повернути на доопрацювання, після цього статус замовлення зміниться на «returned», працівник буде зобов’язаний виправити недоліки вказані замовником, та відправити йому результати заново; в разі необхідності замовник матиме змогу продовжити дедлайн, щоб в працівника було достатньо часу на виправлення недоліків;
- відхилити роботу, в цьому випадку статус замовлення становить «pending», а проєкт повертається на дошку об’явлень; для працівника, що виконував роботу, проєкт зараховується як провалений.

Якщо працівник не вислав роботу до кінцевого строку, то замовнику пропонується вибір наступних варіантів:

- відхилити проєкт, після цього замовлення повернеться на дошку об’явлень із продовженим кінцевим строком, проте доплачувати за нього не потрібно;

- продовжити кінцевий термін, та доплатити за додатковий час, витрачений працівником на проєкт.

Окрім публікації замовлень, замовнику також доступні наступні функціональні можливості:

- перегляд опублікованих замовлень та їхніх статусів;
- перегляд інформації по кожному із замовлень, включаючи результати роботи, вислані працівником;
- відхилення, повернення на доопрацювання чи схвалення виконаної працівником роботи;
- перегляд та редагування профілю;
- перегляд здійснених платіжних операцій;
- чат з працівниками, які виконують опубліковані проєкти.

### *3.1.2. Функціональність для фрілансера*

Зареєструвавшись на сайті, користувачу присвоюється мінімальний ранг та надається доступ до облікового запису, в якому в нього є можливість:

- переглядати статистику (активність, успішність виконання проєктів, доходи);
- переглядати та редагувати профіль;
- переглядати свій ранг, кількість роботи, яку необхідно виконати, щоб піднятись на вищий ранг, та кількість провалених проєктів, за якими слідує пониження рангу або видалення облікового запису із платформи;
- переглядати на дошці об’явлень замовлення від клієнтів відповідного рангу;
- налаштовувати фільтрацію проєктів за рангом та ключовими словами;

- подавати на будь-яке замовлення, доступне на дошці об’явлень, у випадку, якщо користувач не має активних замовлень, тобто не виконує роботу для якогось замовника; в іншому разі, йому необхідно спочатку завершити або відхилити замовлення, на яке він вже подав заявку (у випадку відхилення проєкт буде рахуватись як провалений, тобто певна кількість таких проєктів призведе до пониження рангу або видалення облікового запису із системи);
- переглядати активний проєкт, та відхиляти його чи відсилати результати роботи замовнику;
- підтримувати зв’язок із замовником, для якого виконується робота.

## 3.2. Опис серверної частини проєкту

Для написання серверної частини проєкту використано мову програмування JavaScript та платформу Node.js [14].

Для запуску серверу та написання кінцевих точок (endpoints) для взаємодії з клієнтською частиною використано Express.js. Даний фреймворк надає можливість налаштування (за допомогою використання методу use()) та реалізації API (написання кінцевих точок для обслуговування HTTP запитів). Виглядає це наступним чином:

Лістинг 1. Приклад налаштування та реалізації API

```
app.use(bodyParser.json({
    limit: '10mb',
    extended: false
}))
app.use(passport.initialize())
```

Для реалізації функціональності до обробника відповідної кінцевої точки необхідно першим аргументом вказати шлях, після чого передати як мінімум одну функцію для обробки вхідних даних та відправки відповіді клієнту. Кількість функцій може бути необмеженою. Після завершення

роботи однієї функції, вона повинна передати сигнал на виконання наступної за рахунок методу next(). Такі проміжні функції називаються «middleware». В кожну таку функцію, окрім «callback» функції (функція, що передається в якості аргументу), також передаються аргументи «request» (містить дані, передані від клієнта) та «response» (використовується для надсилання відповіді на запит клієнта).

Виникнення помилок в коді може призвести до падіння сервера. Для того щоб запобігти цьому, всі помилки (непередбачувані та навмисно створені у випадку неправильних даних клієнта, спробі доступу до заборонених ресурсів тощо) необхідно оброблювати. Звісно, можливо просто використовувати try / catch блок у всіх частинах коду, де можливе виникнення помилки, та передавати клієнту повідомлення про виникнення помилки за допомогою методу request.json(). Проте даний спосіб робить код менш читабельним та виникає ризик виникнення непередбачуваних помилок в місцях, не обгорнутих try / catch блоком. Для того, щоб ловити всі помилки, в Express існує спосіб автоматичної обробки всіх помилок:

Лістинг 2. Приклад налаштування API на обробку помилок

```
app.use((err, req, res, next) => {
    res.json({ message: 'Something went wrong :(' })
})
```

Передавши у метод use() функцію з чотирма аргументами, можливо відловлювати всі помилки, які були кинуті в кінцевих точках. Перший аргумент буде являтись помилкою, що була кинута. Таким чином можна уникнути виникнення непередбачуваних ситуацій та реалізувати централізовану функціональність для обробки помилок. Проте в такого способу є один недолік – помилки можуть ловитись тільки ті, які були кинуті в синхронних функціях. В більшості випадків при написанні функціональності використовуються асинхронні функції (для здійснення всередині них таких асинхронних операцій, як читання / запис бази даних). А для того, щоб помилки ловились вищевказаною функцією, їх необхідно

передавати в метод next(). Таким чином без try / catch блоку в будь-якому випадку не обійтись. Проте існують поміжні пакети, такі як express-promise-router, в яких функціональність передачі помилок в асинхронних функціях в метод next() реалізована автоматично. Для таких цілей і було використано даний пакет. Таким чином, всі помилки ловляться в одному місці, а відповідне повідомлення передається клієнту. Для зручності, було реалізовано класи, наслідувані від Error класу, щоб підвищити читабельність коду:

Лістинг 3. Приклад класу помилки

```
class BadRequest extends Error {
    constructor(message) {
        super()
        this.statusCode = 400
        this.message = message
    }
}
```

Для взаємодії з базою даних було використано Mongoose ODM [15]. Для підключення до бази даних використовується метод connect(), а для взаємодії з конкретною моделлю – model(), в яку передається назва моделі та схема. Опис моделі бази даних наведено в п.3.4.

Авторизація користувача реалізована за допомогою пакета Passport [16]. Даний пакет надає можливість реалізації функціональності авторизації як через сесії, так і без їх використання. В даному випадку було використано варіант без сесій. При спробі авторизації користувач передає електронну пошту та пароль, пароль, поєднаний із серверною сіллю, шифрується за допомогою одного з шифрувальних методів, які надаються пакетом Crypto. Після цього здійснюється пошук в базі даних за електронною поштою та зашифрованим паролем. У випадку, якщо користувач був знайдений, генерується токен за допомогою пакету Jsonwebtoken, який передається клієнту. Отримавши токен, клієнт має змогу отримати доступ до ресурсів авторизованого користувача шляхом посилання відповідних запитів разом із токеном. В кожній кінцевій точці,

до якої повинні мати доступ тільки авторизовані користувачі, викликається «middleware», в якому перевіряється токен, переданий клієнтом, на валідність. Якщо токен валідний, надається доступ до відповідних ресурсів, в іншому випадку повертається статус 401.

Для реалізації функціональності збереження та витягування даних великих об’ємів (файли, зображення) було використано Aws-sdk [17] для взаємодії зі сховищем Amazon. На самому сайті є можливість створити так звані «buckets» (сховище для зберігання файлів), для кожного з яких прописується рівень доступу. Тому було створено два сховища – один для аватарів (з публічним доступом) та один для приватних файлів (з обмеженим доступом). Для доступу до сервісу S3, необхідно передати «access key» та «secret key». Проте для доступу до публічних файлів достатньо мати лише посилання на цей файл у сховищі Amazon, тому при поверненні аватарів користувачам, нема необхідності витягувати зображення на сервері, та передавати їх у форматі base64 клієнту, достатньо просто передати посилання на зображення.

Для модульного та інтеграційного тестування серверної частини проєкту було використано пакети Mocha [18], Chai [19] та Supertest (для посилання запитів на сервер). Також було використано пакет Custom-env, за допомогою якого можна вказувати різні значення для змінних в .env, за рахунок чого можна використовувати різні бази даних та порти для тестування та розробки. Таким чином при тестуванні була використана окрема база даних, яка заповнялась даними перед процесом тестування (за допомогою методів before / beforeAll). Для опису тестів використовуються методи describe (для опису окремих кейсів) та it (для опису конкретних випадків). Supertest використовується для реалізації функціональності посилання запитів на сервер, та перевірку відповіді. Дану відповідь можна перевірити за допомогою chai.assert.

### 3.3. Опис клієнтської частини проєкту

Клієнтська частина проєкту розроблено на мові програмування JavaScript з використанням бібліотеки React [20].

Кожен уривок HTML разом з JavaScript функціональністю в React реалізовується в окремому компоненті. Компоненти можуть бути функціональними або реалізованими через класи. Класи мають внутрішній стан та методи життєвого циклу. Функціональні компоненти, до моменту виходу React Hooks, були звичайними функціями, які, в основному, приймали props, та повертали відповідний HTML. Проте з виходом React Hooks появилась можливість добавляти внутрішній стан і у функціональні компоненти, а заміною життєвим циклам являється React Effect. Судячи по всьому, React націлений на повний перехід до функціонального програмування, тому функціональні компоненти отримують все більше можливостей. Проте на даний момент React Effect не мають достатньої функціональності для заміни методів життєвого циклу, тому в даному проєкті вирішено притримуватись використання класів у випадках, коли необхідно реалізувати функціональність з використанням внутрішнього стану чи методів життєвого циклу.

Інколи необхідно мати глобальне сховище для зберігання станів, щоб була можливість передавати стани між різними компонентами, які не пов’язані між собою (тобто неможливо передати через  props). Для цього використовується Redux Framework [21], який являється реалізацією Flux архітектури, проте з певними відмінностями (одне сховище, «immutable» оновлення станів тощо).

Стани зберігаються в Redux Store, доступитись до цих даних можна за допомогою селекторів, а змінити за рахунок виклику відповідних методів («actions» обробників), що передають дані в редуктори (reducers), які в залежності від переданих в них «action» та «state» оновлюють сховище даних.

За допомогою Redux Saga [22] є можливість винести всю асинхронну функціональність (взаємодія із серверною частиною) поза компоненти, що робить код більш читабельним. При використанні Redux Saga використовуються функції-генератори, в яких прописується асинхронна функціональність, та які викликаються при виклику відповідного «action» обробника.

При написанні користувацького інтерфейсу, програміст часто стикається з однотипними завданнями, такими як функціональність форм. Для автоматизації процесів керування станами полів у формах, існує фреймворк Redux Form [23], який виносить та управляє всіма станами полів у формі, та повертає готовий об’єкт при натисненні на кнопку типу «submit».

Для написання стилів до кожного компоненту використовується CSS. Проте з виходом таких препроцесорів, як SASS [24], процес написання стилів став простішим та більш систематизованим. Появились такі можливості, як вкладені стилі, змінні, наслідування тощо, що спрощує процес створення дизайну та робить код більш читабельним.

В системі існують кілька окремих ролей, та відповідно окремих користувацьких інтерфейсів для цих ролей. Так як функціональні можливості для кожної ролі можна фактично вважати окремою бізнес логікою, було вирішено повністю розділити ролі та код так, щоб процес написання функціональності до однієї ролі був як написанням окремого повноцінного застосунку. Тому з метою покращення продуктивності застосунку, було використано пакет @loadable/component, який надає можливість довантажувати компоненти тільки в разі їх необхідності. Таким чином, застосунок використовує компоненти тільки тієї ролі, яка присвоєна користувачу.

На рис. 14 зображена структура клієнтської частини проєкту. В директорії actions описуються всі Redux Actions, в reducers – редуктори. У

файлі actionCreator.js описуються допоміжні функції для створення «actions» обробників. Таким чином спрощується процес їх створення та підвищується читабельність коду.

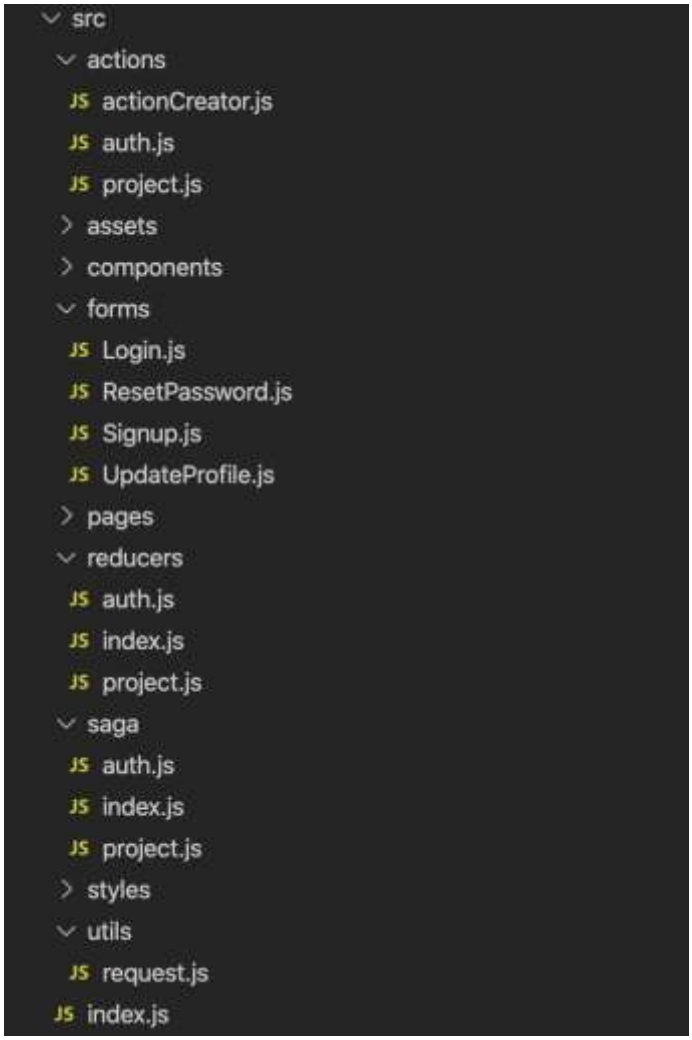



Рис. 14. Структура клієнтської частини проєкту

Лістинг 4. Приклад створення «actions» обробників

```
import { defaultActionCreator } from './actionCreator'

export const LOGIN = 'LOGIN'
export const LOGIN_SUCCESS = `${LOGIN}_SUCCESS`
export const LOGIN_FAIL = `${LOGIN}_FAIL`
export const RESET_LOGIN_DATA = `RESET_${LOGIN}_DATA`

export const login = defaultActionCreator(LOGIN, 'data')
export const loginSuccess = defaultActionCreator(LOGIN_SUCCESS,
'loginMessage')
export const loginFail = defaultActionCreator(LOGIN_FAIL, 'loginError')
export const resetLoginData = defaultActionCreator(RESET_LOGIN_DATA)
```

В директорії saga описується вся асинхронна функціональність, тобто процес взаємодії із серверною частиною проєкту.

В директорії Redux Form описані компоненти, які являються формами. Таким чином функціонал контролю станів форм автоматизований та зберігається в Redux, а також основні компоненти не навантажуються лишнім HTML (який виноситься в директорію forms) та JavaScript кодом.

В директорії assets зберігаються всі зображення, в utils – допоміжні функції (в даному випадку тільки request(), яка використовується Redux Saga для посилання HTTP запитів на сервер), в styles – SCSS стилі.

В директорії pages описується функціональність до кожної сторінки веб-застосунку. Дана директорія (як і customers та styles) розділяє функціонал для різних ролей по різним директоріям (рис. 15).

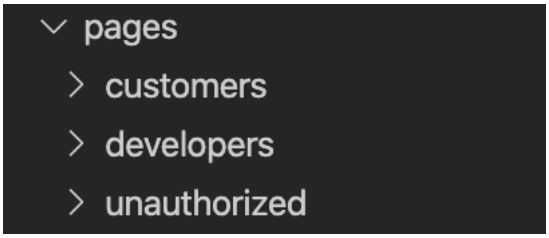



Рис. 15. Розділення функціональності ролей по різним директоріям

Компоненти, описані в директорії pages, описують загальний функціонал та вигляд сторінки, а для опису окремих частин сторінки, використовуються компоненти з директорії components.

### 3.4. База даних

ER діаграма бази даних зображена в Додатку 1. Структура бази даних.

Опис полів моделі Customer:

- name – ім’я користувача;

- email – електронна пошта користувача;
- passHash – зашифрований пароль;
- avatar – посилання на аватар користувача в сховищі Amazon.

Опис полів моделі Developer:

- name – ім’я користувача;
- email – електронна пошта користувача;
- passHash – зашифрований пароль;
- avatar – посилання на аватар користувача в сховищі Amazon;
- rank – ранг фрілансера (можливі значення: F, E, D, C, B, A або S);
- banned – поле логічного типу, яке вказує, чи має користувач доступ до ресурсів (становиться позитивним, коли користувач найнижчого рангу досягнув ліміту завалених проєктів);
- failedProjects – масив завалених проєктів, де id – ідентифікатор проєкту, applied – дата подачі на проєкт, closed – дата закриття проєкту;
- succeedProjects – масив успішно завершених проєктів, де id – ідентифікатор проєкту, applied – дата подачі на проєкт, closed – дата закриття проєкту;
- history – масив проєктів, виконаних працівником на попередніх рангах (адже при зміні рангу масиви успішних та провалених проєктів очищаються), де id – ідентифікатор проєкту, applied – дата подачі на проєкт, closed – дата закриття проєкту, status – статус проєкту (завалений чи успішно завершений);
- searchFilters – фільтри для пошуку проєктів на дошці об’явлень, де rank – ранг проєктів (якщо поле не вказано, здійснюється пошук за всіма доступними рангами, в іншому випадку виключно за вказаним), а tags – ключові слова.

Опис полів моделі Project:

- name – назва проєкту;
- description – опис проєкту, дане поле являється JSON об’єктом, конвертованим у формат String, яке зберігає в собі опис проєкту зі всіма форматуваннями (вирівнювання тексту, курсив тощо).
- attachments – масив посилань на прикріплені до проєкту файли, які зберігаються у сховищі Amazon;
- budget – розмір бюджету, оплачений клієнтом;
- deadline – кінцевий строк подачі результатів роботи фрілансера;
- owner – ідентифікатор замовника, що опублікував замовлення;
- developer – ідентифікатор фрілансера, що виконує роботу;
- rank – ранг, для якого доступний даний проєкт;
- status – статус замовлення (pending, active, returned, failed або closed);
- created – дата публікації замовлення;
- applied – дата початку роботи фрілансера над проєктом;
- closed – дата завершення проєкту;
- isOverdue – поле логічного типу, яке вказує на те, чи пройшов кінцевий строк, вказаний замовником;
- milestones – масив етапів розроблення проєкту, де title – заголовок, budget – частина з головного бюджету, виділена на розробку даного етапу, description – опис роботи, яку необхідно виконати (аналогічно полю description, загальному опису проєкту), deadline – кінцевий строк відправки результатів роботи по даному етапі, finished – логічне значення, що вказує, чи завершений даний етап.

Опис полів моделі Chat:

- participants – масив ідентифікаторів учасників чату.

Опис полів моделі Message:

- chatId – ідентифікатор чату;
- owner – ідентифікатор відправника повідомлення;
- message – текстове повідомлення;
- read – дата, коли повідомлення було прочитане співбесідником;
- sent – дата відправки повідомлення.

# 4. АНАЛІЗ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1. Опис інтерфейсу користувача

На головній сторінці відображається загальна інформація про веб-застосунок та кнопки авторизації / реєстрації.

При натисненні на кнопку реєстрації, користувача переадресовує на сторінку, на якій йому в першу чергу необхідно вибрати свою роль (рис. 17).


Рис. 17. Вибір ролі користувача

Після вибору ролі відкривається форма для заповнення даних (рис. 18).


Рис. 18. Заповнення даних користувача

Після реєстрації користувача переадресовує на головну сторінку.

### *4.1.1. Опис інтерфейсу для замовника*

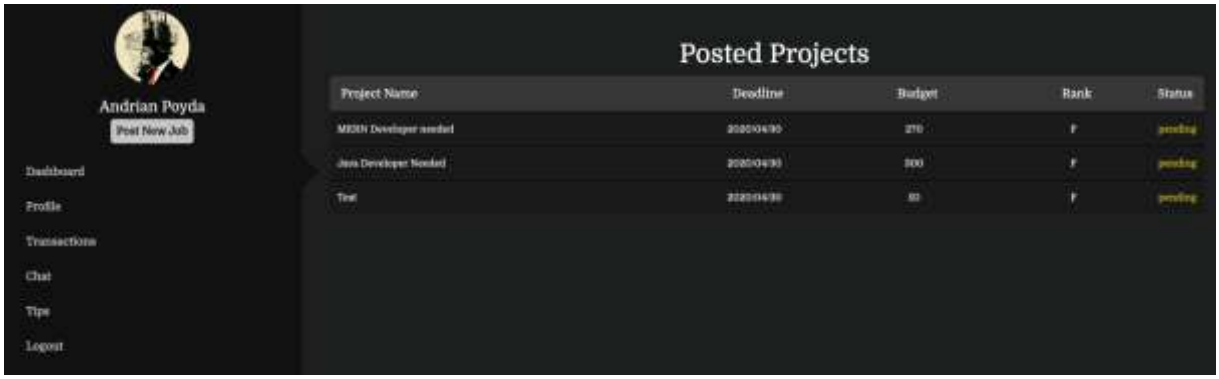



Рис. 19. Головна сторінка замовника

На кожній сторінці відображається меню з аватаром, кнопкою для публікації нового замовлення, кнопкою для виходу з облікового запису та панеллю для навігації по сторінкам.

На головній сторінці відображається список опублікованих замовлень (рис. 19) із наступною інформацією:

- назва проєкту;
- кінцевий строк;
- бюджет, виділений на розробку проєкту;
- ранг працівника, який розроблятиме продукт;
- статус замовлення.

При натисненні на будь-який проєкт зі списку, випливає вікно з деталями проєкту (рис. 20).


Рис. 20. Деталі проєкту

Для публікації нового замовлення, користувачу необхідно натиснути на кнопку «Post New Job», що розташована в меню, після чого з’явиться вікно для заповнення даних (рис. 21).

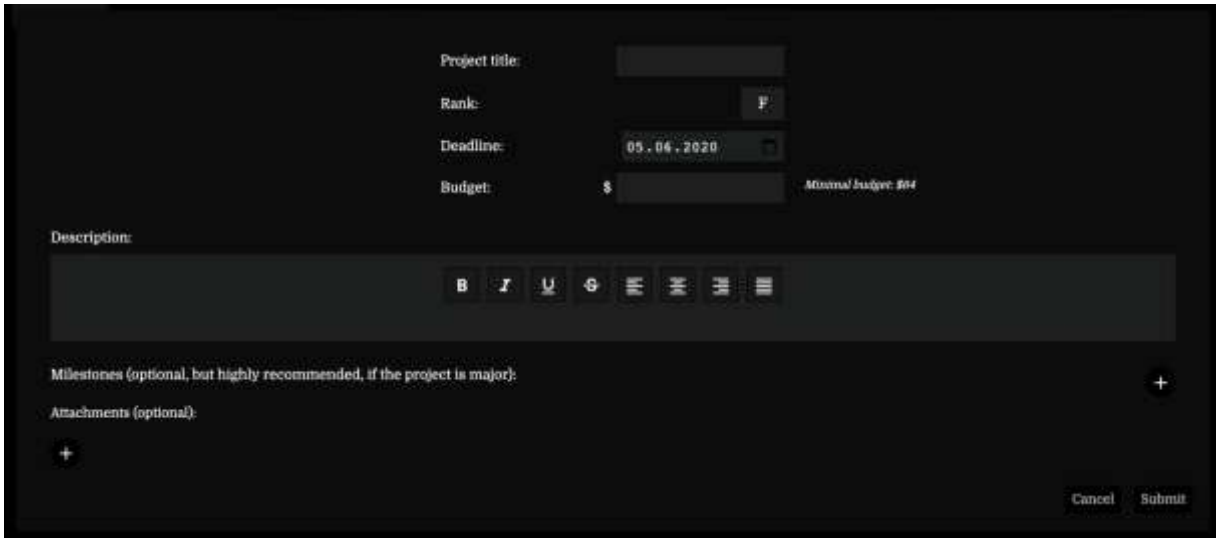



Рис. 21. Форма для публікації нового проєкту

Окрім заповнення загальної інформації про проєкт, є можливість його розділення на кілька етапів. Для цього існує розділ «Milestones», в

якому можна добавити необмежену кількість блоків з інформацією про кожен етап (рис. 22).


Рис. 22. Етапи розроблення проєкту

Після заповнення всіх полів та здійснення оплати на суму, вказану в полі «Budget», вікно закриється, а замовлення опублікується на дошці об’явлень відповідного рангу.

На сторінці профілю відображається інформація про користувача та спосіб оплати (рис. 23), на якій є можливість оновити спосіб оплати, ім’я, електронну пошту, чи пароль користувача.

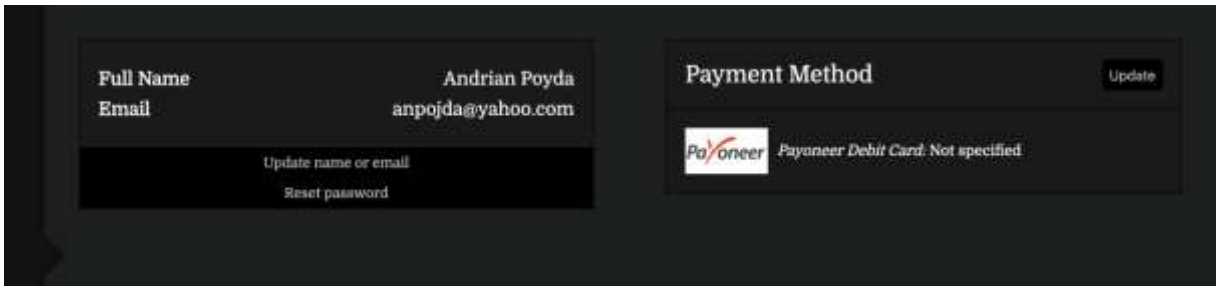



Рис. 23. Сторінка профілю

На сторінці транзакцій є можливість переглянути список проведених транзакцій з інформацією про замовлення (рис. 24).

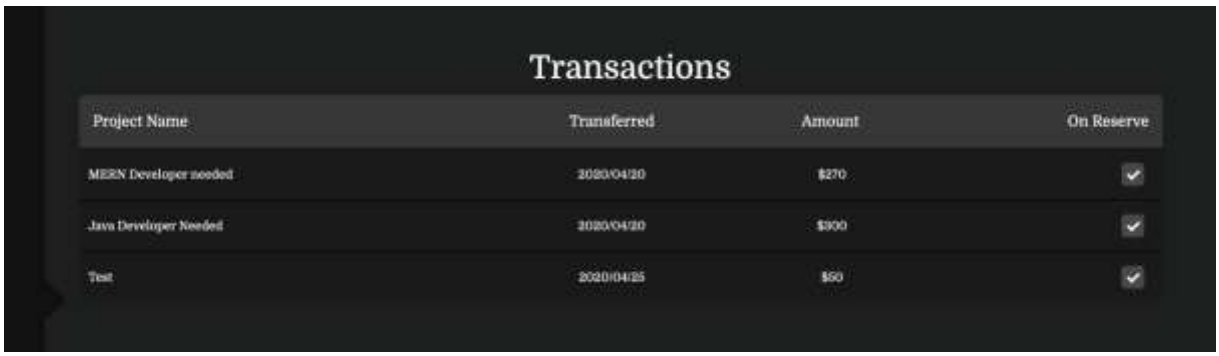



Рис. 24. Сторінка транзакцій

На сторінці чату відображається чат для підтримки зв’язку з працівниками (рис. 25).

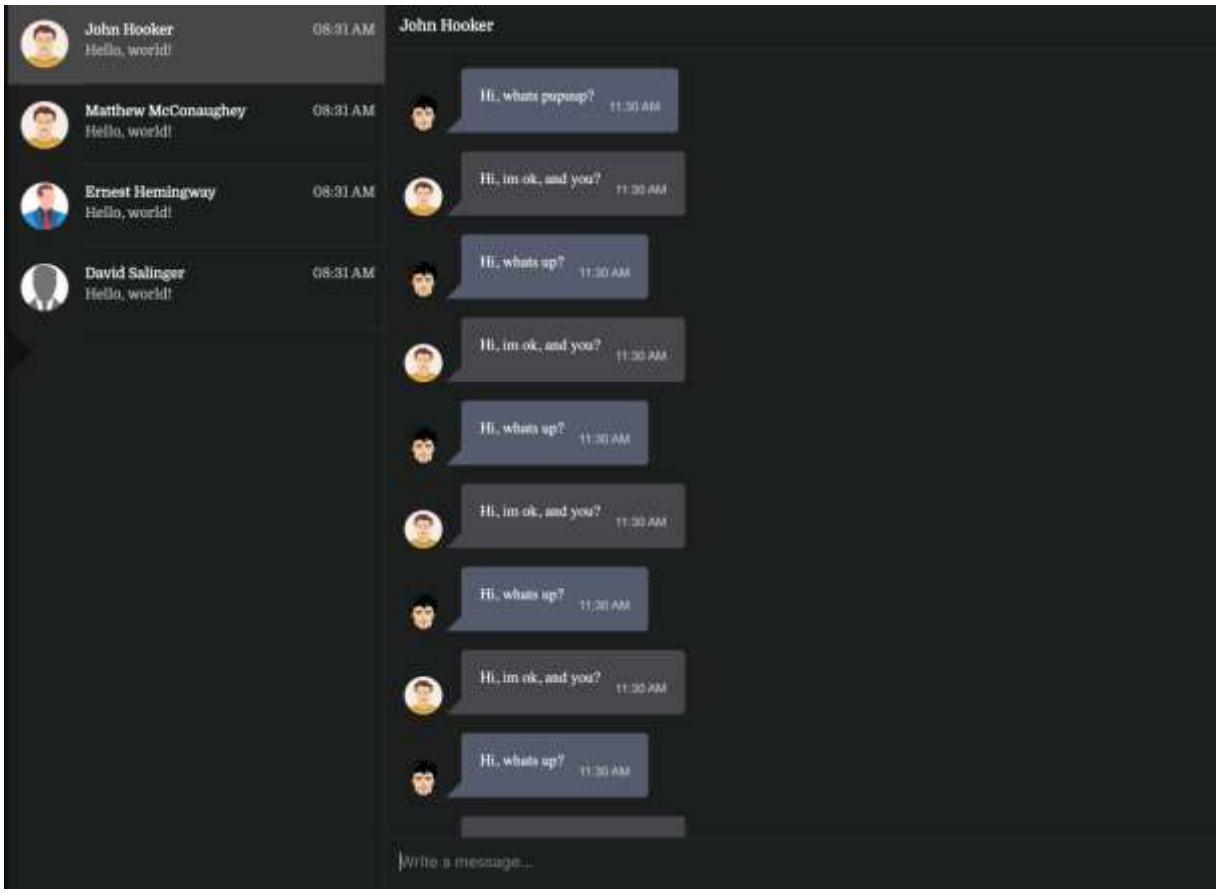



Рис. 25. Чат

На сторінці Tips розташовані рекомендації для користувача щодо користування платформою.

### *4.1.2. Опис інтерфейсу для працівника*


Рис. 26. Головна сторінка працівника

На кожній сторінці відображається меню з аватаром, рангом користувача, кнопкою для виходу з облікового запису та панеллю для навігації по сторінкам.

На головній сторінці відображається прогрес користувача:

- існуючі ранги та пройдені серед них працівником;
- відсоток успішно завершених проєктів до загальної кількості, за якою слідує підвищення рангу;
- відсоток провалених проєктів до загальної кількості, за якою слідує пониження рангу або видалення облікового запису із системи.

Також на сторінці є кнопки, при натисненні на які появляється вікно (рис. 27) з наступною інформацією:

- статистика активності працівника та його доходів;
- статистика успішності проєктів;
- профіль користувача з можливістю його редагування;

- спосіб оплати;
- рекомендації щодо користування платформою.

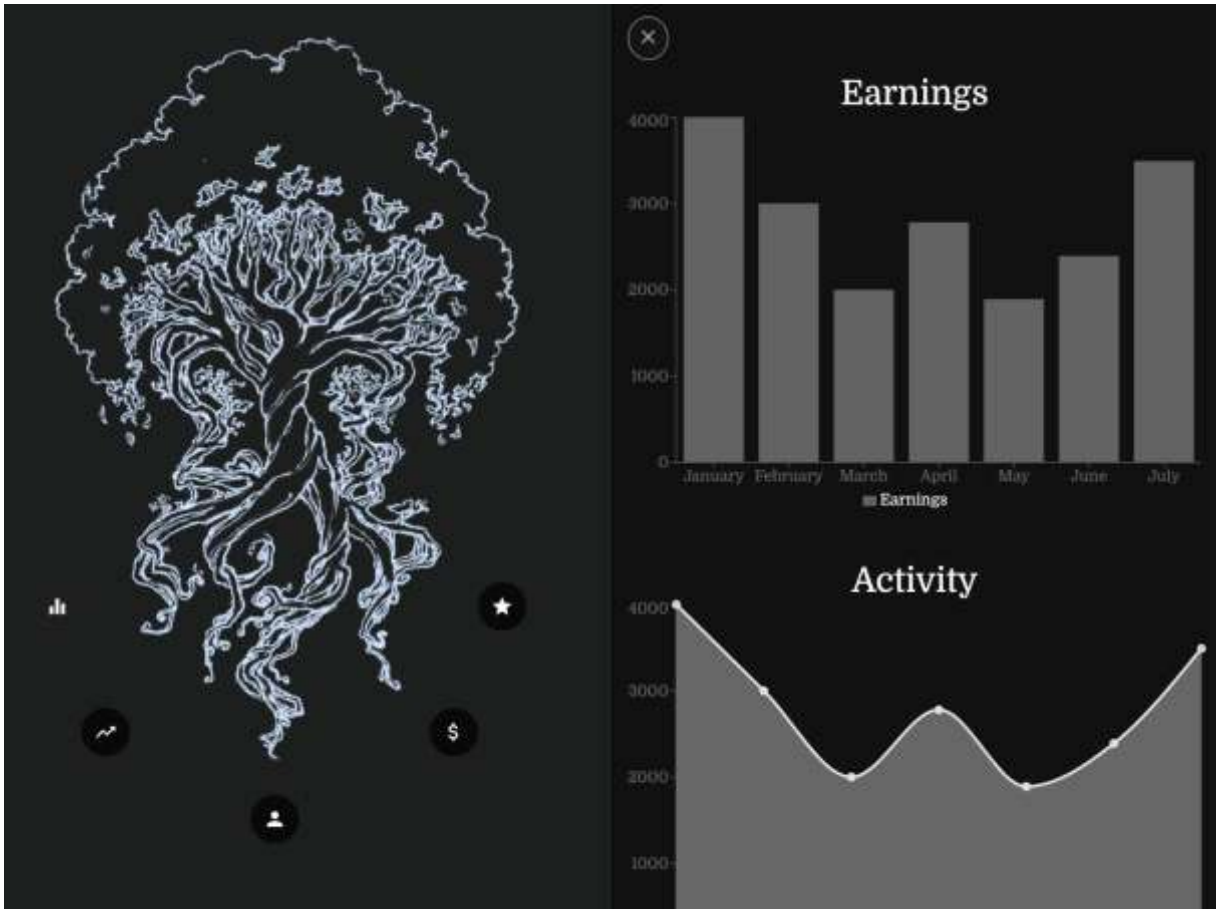



Рис. 27. Вікно з відображенням інформації на головній сторінці

На сторінці проєкту відображається детальна інформація про проєкт, на який подав працівник, та можливість його відхилення чи подачі результатів роботи (рис. 28).

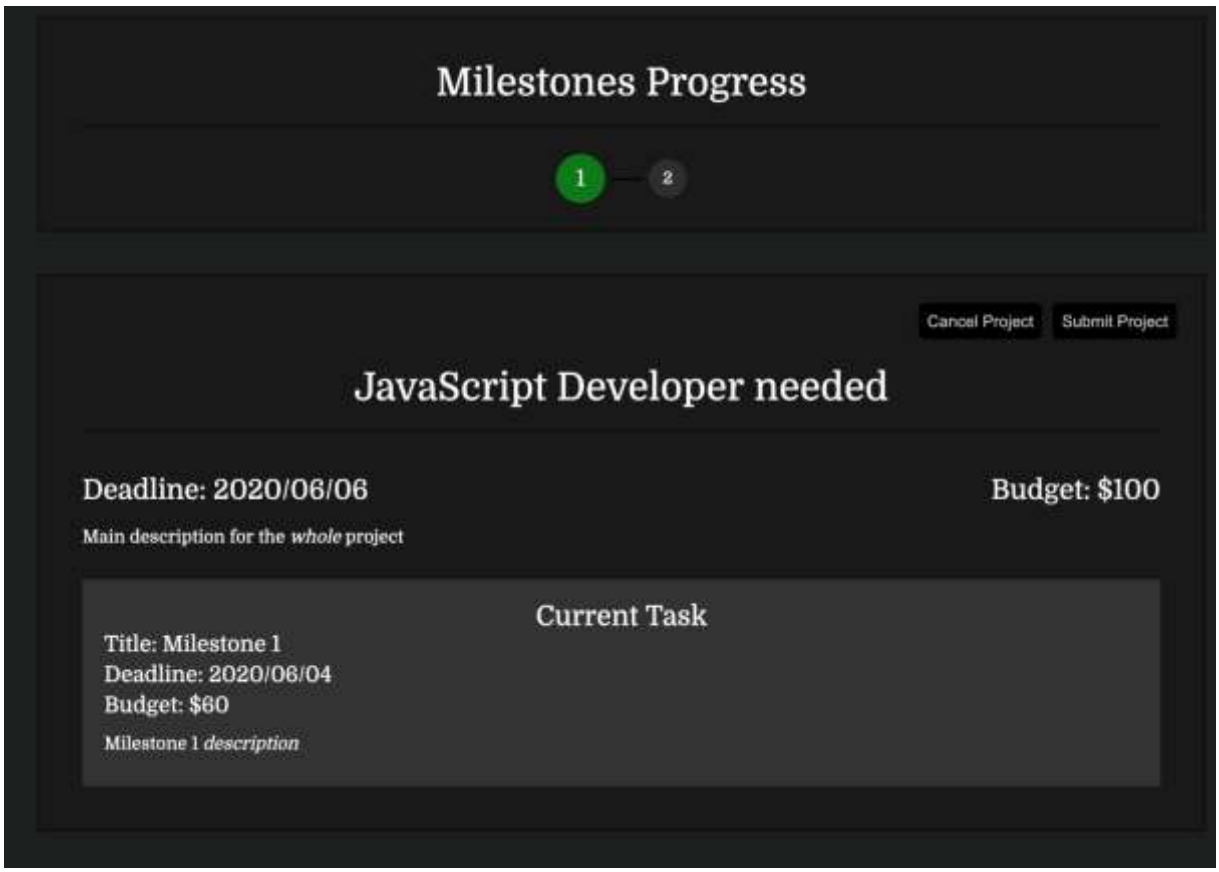



Рис. 28. Сторінка проєкту, над яким працює фрілансер

На сторінці дошки об’явлень відображаються вільні замовлення відповідного рангу (рис. 29).

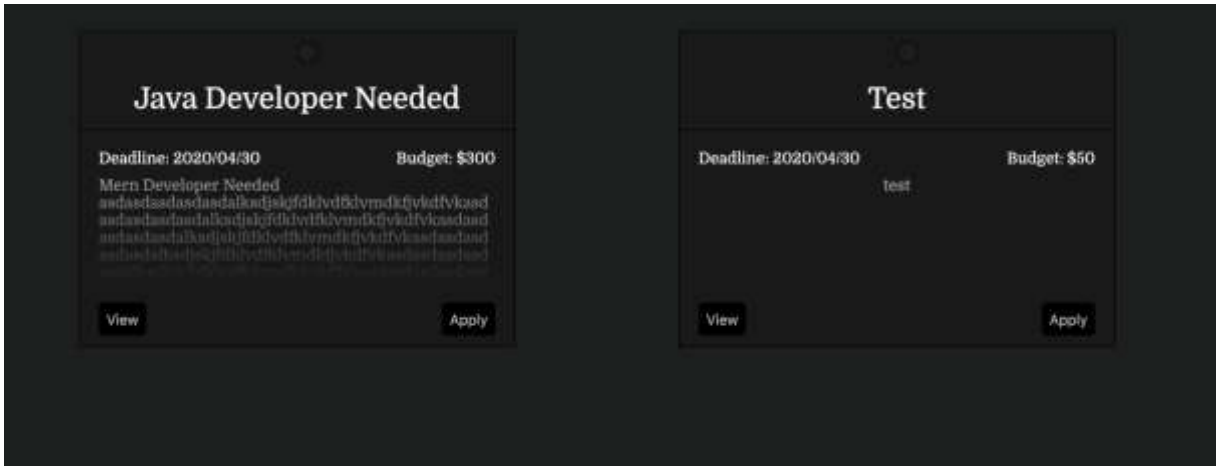



Рис. 29. Список опублікованих замовлень

При натисненні на кнопку View відкривається вікно з детальною інформацією про проєкт (рис. 30).


Рис. 30. Детальна інформація про проєкт на дошці об’явлень

В правому верхньому кутку сторінки є кнопка, при натисненні на яку відкриється вікно для налаштування фільтрації проєктів (рис. 31).


Рис. 31. Налаштування фільтрації проєктів

На сторінці чату відображається чат для підтримки зв’язку із замовниками.

## 4.2. Тестування додатка

Для серверної частини додатка було проведене інтеграційне тестування за допомогою бібліотек mocha, chai та supertest. Були прописані

тести для кожної кінцевої точки з метою покриття тестами всієї функціональності сервера.

Лістинг 5. Приклад написання інтеграційних тестів

```
describe('Log in', () => {
    it('should return 400 if credentials are missing', (done) => {
        agent
            .post('/api/login')
            .send({})
            .expect(400, done)
    })

    it('should return 400 if email is incorrect', (done) => {
        agent
            .post('/api/login')
            .send({ email: 'not@exist.com', password: '123' })
            .expect(400, done)
    })

    it('should return 400 if password is incorrect', (done) => {
        agent
            .post('/api/login')
            .send({ email: 'test@test.com', password: '123' })
            .expect(400, done)
    })

    it('should return 200, role and token', (done) => {
        agent
            .post('/api/login')
            .send({ email: 'test@test.com', password: '12345678' })
            .expect(res => {
                assert.strictEqual(res.body.role, 'customer')
                assert.isTrue(res.body.token && res.body.token.length > 0)
            })
            .expect(200, done)
    })
})
```

Для клієнтської частини веб-додатка було проведене ручне тестування.

### *4.2.1. Реєстрація користувача*

Кроки до відтворення:

- перейти на сторінку реєстрації;
- ввести обов’язкові дані;
- натиснути кнопку Submit.

Очікувані результати:

- створюється новий обліковий запис, користувач автоматично авторизується.

### *4.2.2. Авторизація користувача*

Кроки до відтворення:

- перейти на сторінку авторизації;
- ввести електронну пошту та пароль;
- натиснути кнопку Submit.

Очікувані результати:

- користувач успішно авторизується в системі.

### *4.2.3. Публікація замовлення (роль – замовник)*

Кроки до відтворення:

- відкрити вікно публікації замовлення (користувач з роллю замовника);
- ввести всі обов’язкові дані;
- натиснути кнопку Submit;
- здійснити оплату.

Очікувані результати:

- замовлення появляється на дошці об’явлень працівників відповідного рангу;
- замовлення появляється в списку опублікованих проєктів на головній сторінці замовника;
- проведена оплата додається в список транзакцій.

### *4.2.4. Подання на заявки проєкт (роль – працівник)*

Кроки до відтворення:

- перейти на сторінку дошки об’явлень;

- натиснути на кнопку Apply відповідного замовлення.

Очікувані результати:

- замовлення з’являється на сторінці Project працівника;
- статус замовлення змінюється на «active»;
- на сторінці чату появляється можливість зв’язатись із замовником чи працівником;
- замовлення пропадає з дошки об’явлень.

### *4.2.5. Відхилення проєкту (роль – працівник)*

Кроки до відтворення:

- перейти на сторінку проєкту;
- натиснути на кнопку Cancel Project.

Очікувані результати:

- статус замовлення змінюється на «pending»;
- замовлення пропадає зі сторінки Project відповідного користувача, та з’являється на сторінці дошки об’явлень;
- проєкт зараховується для працівника як провалений.

### *4.2.6. Подача результатів проєкту (роль – працівник)*

Кроки до відтворення:

- перейти на сторінку проєкту;
- натиснути на кнопку Submit Results;
- написати звіт;
- прикріпити документи;
- натиснути на кнопку Submit.

Очікувані результати:

- документи відправляються замовнику;
- статус замовлення становиться «on review»;

- операції відміни чи відправки результатів становляться недоступними для працівника.

### *4.2.7. Ухвалення виконаної працівником роботи (роль – замовник)*

Кроки до відтворення:

- перейти на головну сторінку;
- відкрити відповідний проєкт;
- переглянути надіслані документи;
- натиснути на кнопку Approve.

Очікувані результати:

- якщо проєкт не розділений на кілька етапів, або поточний етап являється останнім:
  - проєкт зараховується для працівника як успішно завершений;
  - статус замовлення змінюється на «closed»;
  - кошти із резервного фонду переводяться працівнику;
- якщо проєкт розділений на кілька етапів, та поточний етап не є останнім:
  - статус замовлення змінюється на «active»;
  - операції відміни та відправки результатів знову стають доступними для працівника.

### *4.2.8. Повернення роботи (роль – замовник)*

Кроки до відтворення:

- перейти на головну сторінку;
- відкрити відповідний проєкт;
- переглянути надіслані документи;
- натиснути на кнопку Return Project.

Очікувані результати:

- проєкт повертається працівнику, щоб він міг виправити недоліки, вказані замовником.

#### *4.2.9. Відхилення роботи (роль – замовник)*

Кроки до відтворення:

- перейти на головну сторінку;
- відкрити відповідний проєкт;
- переглянути надіслані документи;
- натиснути на кнопку Decline Project.

Очікувані результати:

- статус замовлення змінюється на «pending»;
- замовлення пропадає зі сторінки Project відповідного користувача, та з’являється на сторінці дошки об’явлень;
- проєкт зараховується для працівника як провалений.

### 4.3. Шляхи подальшого розвитку

Після проведення аналізу розробленого веб-застосунку, сформовано шляхи подальшого розвитку проєкту, які полягають в розширенні функціональних можливостей, а також в деякій зміні моделі роботи системи рангів.

На даний момент найнижчий ранг працівників виконують роботу за відповідну суму коштів. Пропонується зробити найнижчий ранг безкоштовним, що буде певним випробувальним терміном для працівників, та можливістю отримати безкоштовно роботу для замовників.

Підвищення рангу на даний момент здійснюється автоматично при завершенні працівником певної кількості успішних проєктів (розраховується, враховуючи час, протягом якого виконувався кожен проєкт). Для більш коректного відбору спеціалістів пропонується ввести наступну функціональність:

- після успішного завершення працівником достатньої кількості проєктів, система вибирає випадковим чином певну кількість працівників, які мають вищий ранг, та посилає їм запрошення провести інтерв'ю;
- працівник проходить всі інтерв'ю, після чого кожен користувач, що проводив інтерв'ю, виносить свій вердикт;
- якщо більшість проголосувало за підвищення рангу працівнику, то ранг підвищується; в протилежному випадку ранг працівника не змінюється, а успішність успішно завершених та провалених проєктів анулюється, після чого працівник починає заново;
- користувачі, що проводили інтерв'ю, і чий вердикт збігся з кінцевим рішенням системи, отримують певні бонуси (наприклад, зменшення кількості проєктів, які необхідно завершити для підвищення рангу).

Також пропонується ввести функціональність верифікації користувача за паспортом чи ідентифікаційним кодом з метою уникнення ситуацій створення користувачем нових облікових записів після його блокування в системі.

Пропонується ввести роль внутрішніх працівників, які будуть користувачами, що відповідають за відбір працівників при їх реєстрації, тобто проводять з ними бесіди, щоб переконатись в достатньому рівні кваліфікації та "soft skills". Також дані користувачі відповідатимуть за аналіз ситуації у випадку відхилення проєкту замовником, та виноситимуть рішення про подальші дії.

Таким чином, в користувачів із цією роллю будуть наступні можливості:

- чат для підтримки зв’язку із користувачами, в тому числі і з потенційними працівниками;
- доступ до всіх ресурсів проєкту, в якому замовник відхилив роботу працівника;
- права на видачу доступу до облікового запису новим користувачам;
- права на винесення кінцевих рішень в ситуаціях завалених проєктів (зараховувати проєкт для працівника як провалений чи ні, повертати кошти замовнику чи повертати замовлення на дошку об’явлень).

# ВИСНОВКИ

Даний дипломний проєкт присвячено створенню веб-додатка для пошуку дистанційної роботи на основі системи рангів.

Потреби замовників та фрілансерів було зібрано та проаналізовано. На основі потреб висунуто критерії оцінювання програмних засобів для проведення порівняльного аналізу існуючих на ринку рішень. Розглянуті аналоги не задовольняють поставленим вимогам у повній мірі, чим обґрунтовується необхідність розроблення додатка для вирішення проблем пошуку та відбору працівників зі сторони замовників, та пошуку роботи зі сторони фрілансерів.

У даній роботі проаналізовано та вибрано найбільш доцільні для даного проєкту технології розробки. Було вирішено використовувати мову програмування JavaScript, та фреймворки Node та React для розроблення серверної та клієнтської частин відповідно. В якості сховища даних було вибрано СКБД MongoDB для збереження та керування даними, та Amazon S3 для збереження громіздких даних, таких як файли чи зображення.

У дипломному проєкті розроблено серверну та клієнтську архітектури додатка, та структуру бази даних. Для початку роботи з додатком передбачена авторизація. Забезпечено розділення функціональності через набір ролей користувачів (працівник та замовник). Замовник має змогу публікувати проєкти, приймати роботи від працівників, здобрювати, відхиляти чи відправляти їх на доопрацювання. Працівник в свою чергу має змогу переглядати всі замовлення, які відповідають його рангу, подавати на будь-яке з них, відхиляти, та відправляти результати роботи замовнику. Також замовник та працівник мають змогу взаємодіяти один з одним через імплементований чат. Реалізована функціональність спрощує процес найму працівників, та дає можливість новим користувачам сайту проявити себе.

## СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

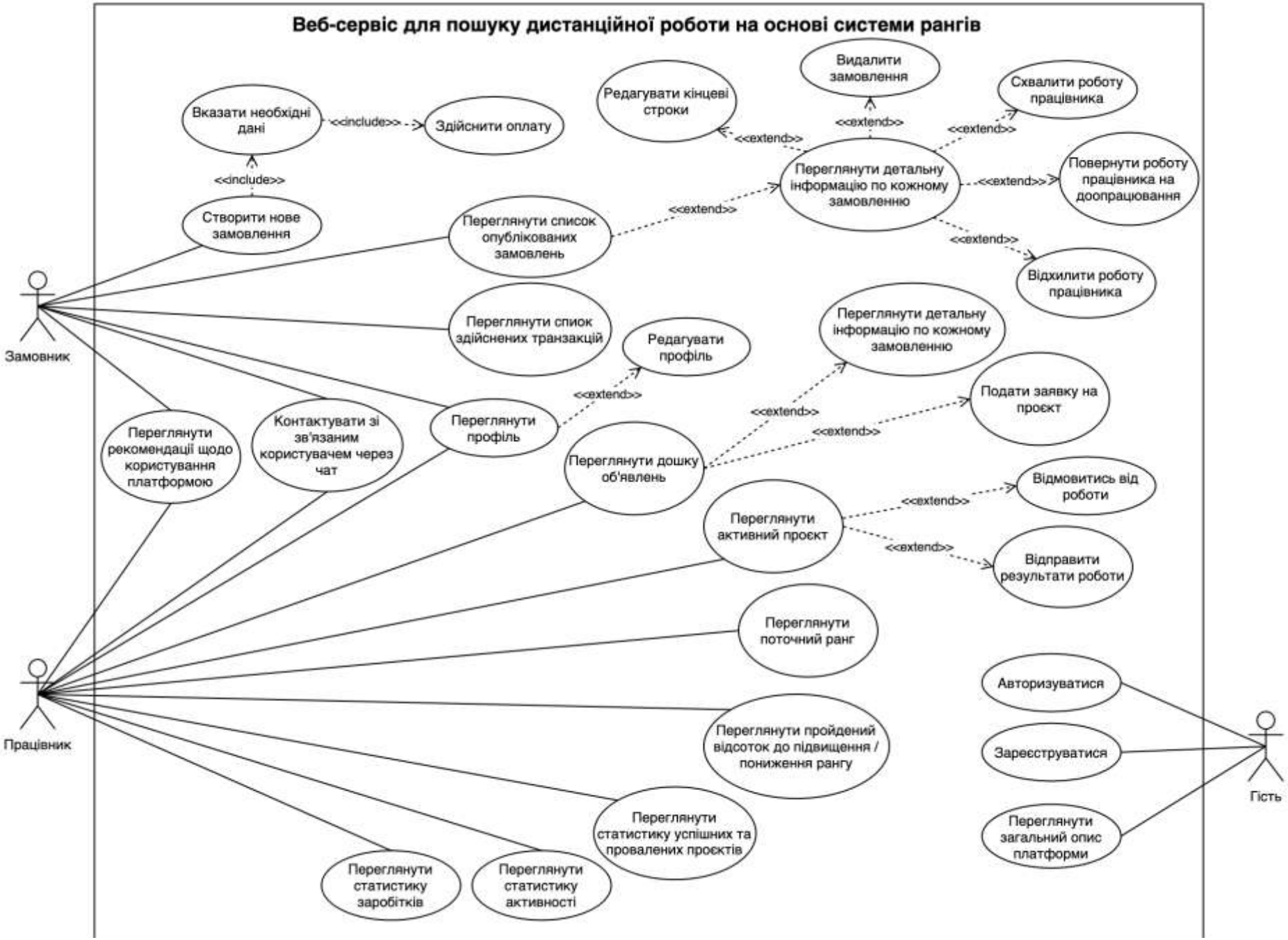

1. Список найкращих платформ для пошуку дистанційної роботи [Електронний ресурс]. Режим доступу – https://loleknbolek.com/birzhi-freelance/.
2. Загальний опис мови програмування JavaScript [Електронний ресурс]. Режим доступу – https://developer.mozilla.org/uk/docs/Web/JavaScript.
3. Порівняння JavaScript фреймворків для розробки клієнтської частини [Електронний ресурс]. Режим доступу – http://mkdev.me/posts/sravnenie-javascript-freymvorkov-vue-js-react-i-angular-2019.
4. Загальний опис JSX [Електронний ресурс]. Режим доступу – https://codeguida.com/post/363.
5. Загальний опис Node.js [Електронний ресурс]. Режим доступу – https://nodejs.org/en/about/.
6. Інформація про всі технології Spring Projects [Електронний ресурс]. Режим доступу – https://spring.io/projects.
7. Мова програмування Python та її фреймворки для веб-розробки [Електронний ресурс]. Режим доступу – https://www.bitdegree.org/tutorials/python-web-development/.
8. Загальний опис СКБД PostgreSQL [Електронний ресурс]. Режим доступу – https://www.postgresql.org/about/.
9. Порівняння СКБД PostgreSQL та MongoDB PostgreSQL [Електронний ресурс]. Режим доступу – https://www.educba.com/mongodb-vs-postgresql/.
10. Переваги та недоліки найбільш популярних СКБД [Електронний ресурс]. Режим доступу – https://www.keycdn.com/blog/popular-databases.
11. Порівняння СКБД MongoDB та DynamoDB [Електронний ресурс]. Режим доступу – https://www.educba.com/mongodb-vs-dynamodb/.

12. Загальний опис СКБД MongoDB [Електронний ресурс]. Режим доступу – https://www.guru99.com/what-is-mongodb.html.
13. Особливості СКБД MongoDB [Електронний ресурс]. Режим доступу – https://www.javatpoint.com/mongodb-features.
14. Технічна документація Node.js [Електронний ресурс]. Режим доступу – https://nodejs.org/docs/latest-v12.x/api/.
15. Технічна документація Mongoose.js [Електронний ресурс]. Режим доступу – https://mongoosejs.com/docs/guide.html.
16. Технічна документація Passport.js [Електронний ресурс]. Режим доступу – http://www.passportjs.org/docs/downloads/html/.
17. Технічна документація Amazon S3 [Електронний ресурс]. Режим доступу – https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html.
18. Технічна документація Mocha [Електронний ресурс]. Режим доступу – https://mochajs.org/.
19. Технічна документація Chai.js [Електронний ресурс]. Режим доступу – https://www.chaijs.com/api/.
20. Технічна документація React.js [Електронний ресурс]. Режим доступу – https://reactjs.org/docs/getting-started.html.
21. Технічна документація Redux.js [Електронний ресурс]. Режим доступу – https://redux.js.org/api/api-reference.
22. Технічна документація Redux Saga [Електронний ресурс]. Режим доступу – https://redux-saga.js.org/.
23. Технічна документація Redux Form [Електронний ресурс]. Режим доступу – https://redux-form.com/8.3.0/docs/api/.
24. Технічна документація Sass Pre-processor [Електронний ресурс]. Режим доступу – https://sass-lang.com/documentation.

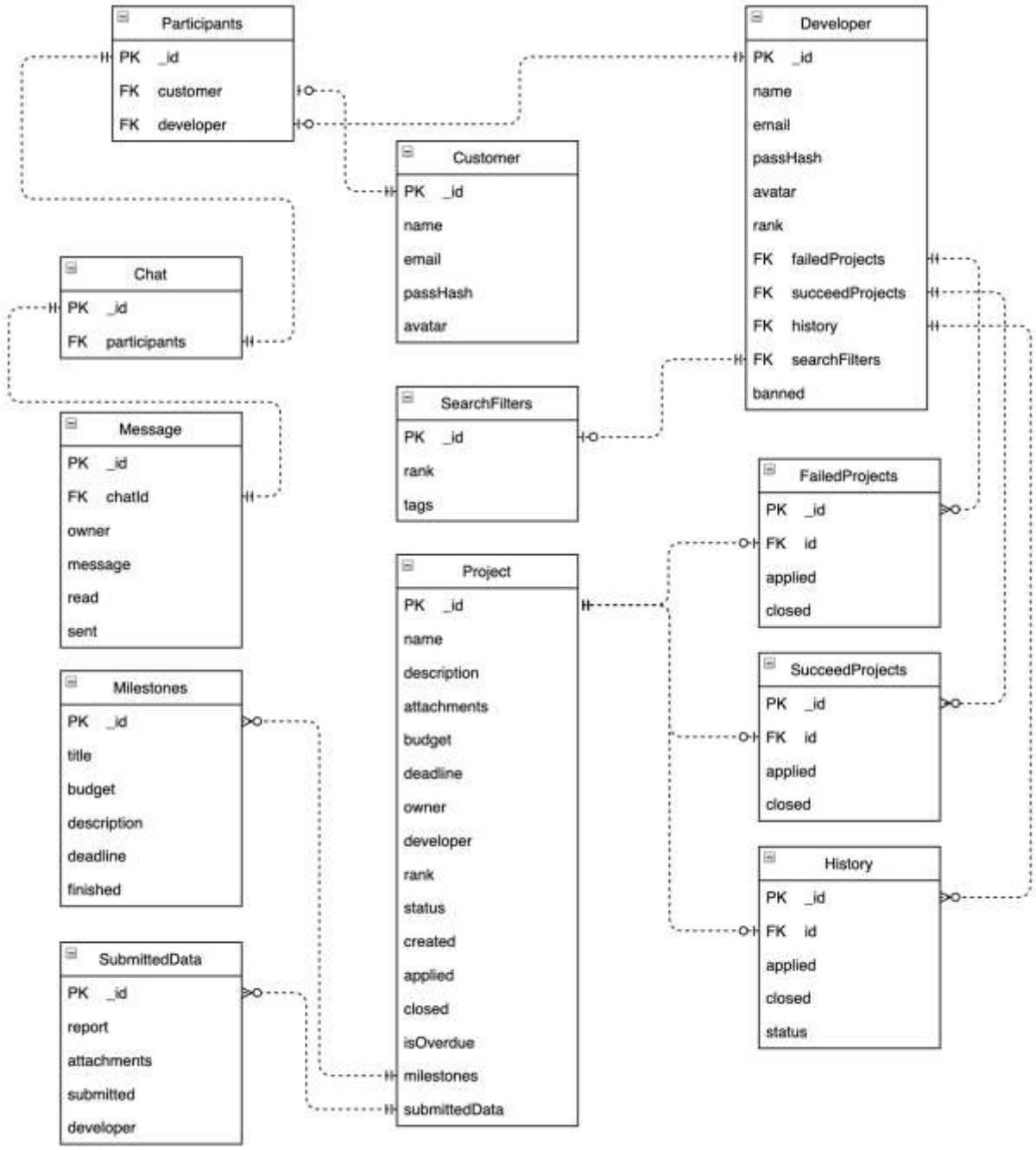
# ДОДАТКИ

**Додаток 1**

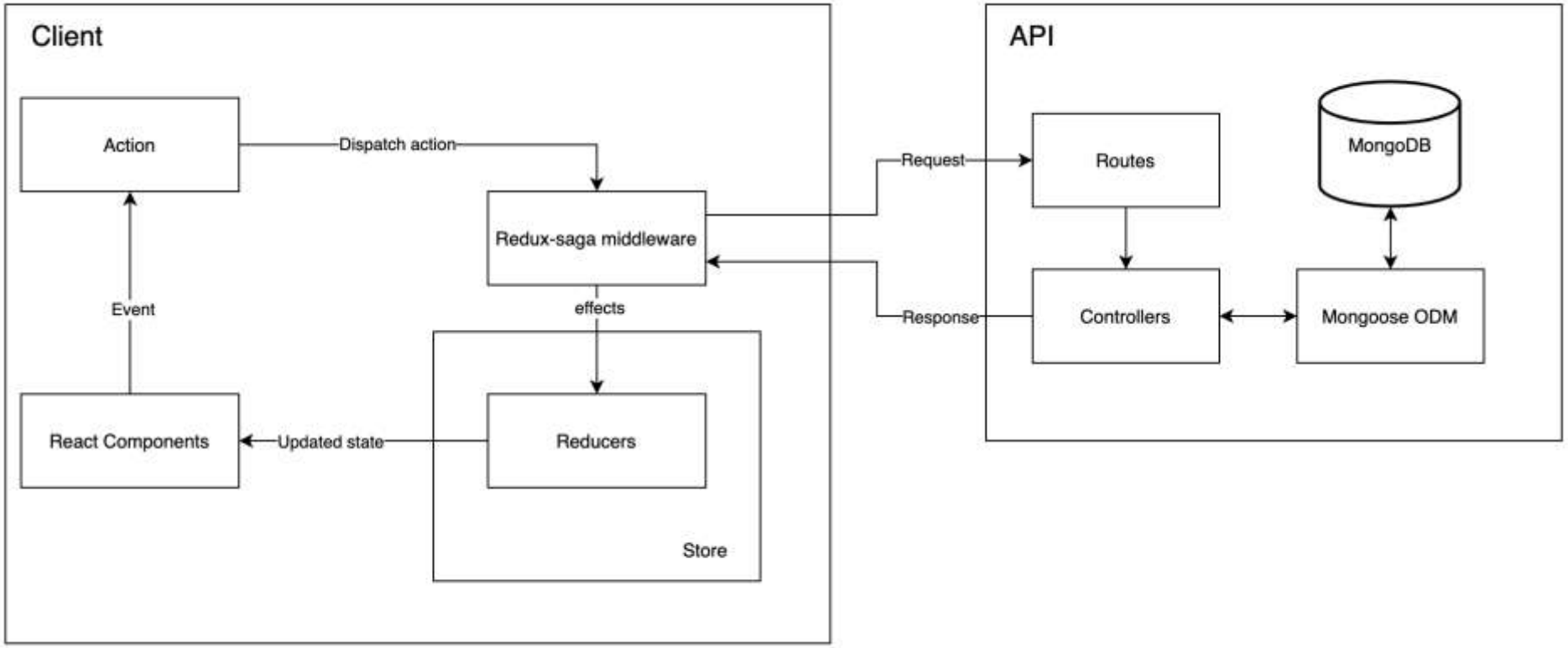
**Копії графічних матеріалів**



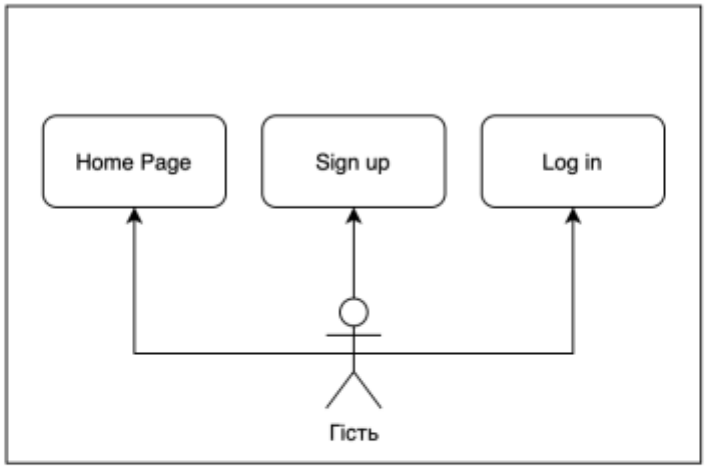

ДП.045440-06-99
Веб-сервіс для пошуку дистанційної роботи на основі системи рангів.
Концептуальний опис поведінки системи. Діаграма прецедентів

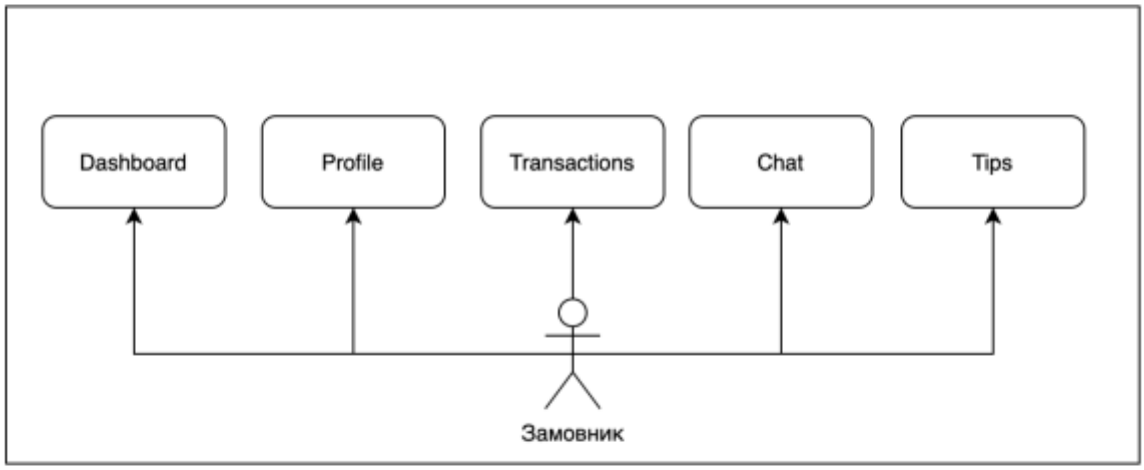


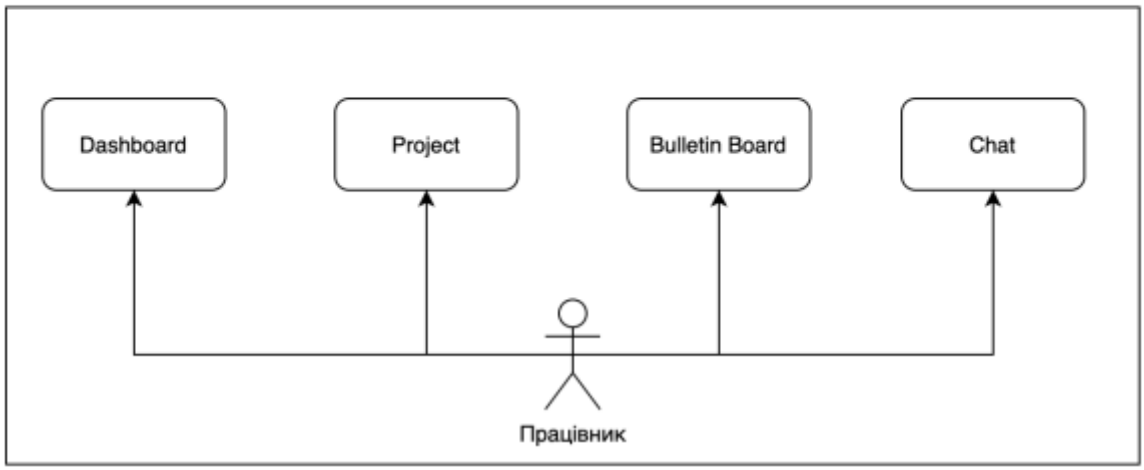

ДП.045440-07-99
Веб-сервіс для пошуку дистанційної роботи на основі системи рангів.
Структура бази даних. ER діаграма




Архітектура веб-додатка
Пойда А.В., група КП-61




Структура веб-сторінок
Пойда А.В., група КП-61

**Додаток 2**
**Лістинг програми**

## index.js

```
const express = require('express')
const app = express()
const bodyParser = require('body-parser')
const passport = require('passport')
const path = require('path')
const mongoose = require('mongoose')
const { handleError } = require('./errors')
const http = require('http').createServer(app)
const { initIo } = require('./utils/chat')
require('custom-env').env(true)

mongoose.connect(process.env.MONGODB_URI, { useNewUrlParser: true })

app.use(bodyParser.json({
    limit: '10mb',
    extended: false
}))
app.use(passport.initialize())

app.use(express.static(path.join(__dirname     +     '/client/codderasil/',
'build')))

require('./authentication').init()

require('./routes').init(app)

app.get('/*', function (req, res) {
    res.sendFile(path.join(__dirname                                        +
'/client/codderasil/build/index.html'))
})

app.use((err, req, res, next) => {
    handleError(err, res)
})

initIo(http)

http.listen(process.env.PORT, () => {
    console.log('server is listening at %s', process.env.PORT)
})

module.exports = { app }
```

## utils/chat.js

```
const Chat = require('../models/chat')
const Message = require('../models/message')
const decodeToken = require('../utils/decode-token')

let io = null

const initIo = (http) => {
    io = require('socket.io')(http)
    listenForMessages()
}

const createNewChat = async (participants) => {
    await Chat.create({
        participants: participants.map(participant => {
            return { id: participant }
```

```
        })
    })
    listenForMessages()
}

const listenForMessages = async () => {
    const chatIds = (await Chat.find()).map(chat => chat._id)
    for (const chatId of chatIds) {
        const nsp = io.of(`/${chatId}`)
        nsp.on('connection', (socket) => {
            socket.on('message', async (data) => {
                const user = await decodeToken(data.token)
                const message = await Message.create({
                    chatId,
                    owner: user._id,
                    message: data.message,
                    read: false,
                    sent: new Date()
                })
                nsp.emit('message', {
                    message,
                    sendingId: data.sendingId
                })
            })
            socket.on('message-viewed', async (data) => {
                const user = await decodeToken(data.token)
                const message = await Message.findById(data.messageId)
                if (user._id !== message.owner) {
                    message.read = true
                    await message.save()
                }
            })
        })
    }
}

const deleteChatByParticipants = async (participants) => {
    const chat = await Chat.findOne({
        participants: {
            $all: participants.map(participant => {
                return { $elemMatch: { id: participant } }
            })
        }
    })
    await Message.deleteMany({ chatId: chat._id })
    await chat.remove()
    listenForMessages()
}

module.exports = {
    initIo,
    createNewChat,
    deleteChatByParticipants
}
```

## utils/decode-token.js

```
const jwt = require('jsonwebtoken')
const config = require('../config')
const Customer = require('../models/customer')
const Developer = require('../models/developer')
```

```
module.exports = (token) => {
    return new Promise((resolve, reject) => {
        jwt.verify(token, config.jwtSecret, async (err, decoded) => {
            if (err) reject(err)

            const userId = decoded.sub
            const role = decoded.role
            if (role !== 'customer' && role !== 'developer') reject()

            try {
                let user
                if (role === 'developer') user = await
Developer.findById(userId).lean()
                else user = await Customer.findById(userId).lean()
                if (!user) reject()
                resolve({ ...user, role })
            } catch (err) {
                reject()
            }
        })
    })
}
```

## utils/decode-token.js

```
const jwt = require('jsonwebtoken')
const config = require('../config')
const Customer = require('../models/customer')
const Developer = require('../models/developer')

module.exports = (token) => {
    return new Promise((resolve, reject) => {
        jwt.verify(token, config.jwtSecret, async (err, decoded) => {
            if (err) reject(err)

            const userId = decoded.sub
            const role = decoded.role
            if (role !== 'customer' && role !== 'developer') reject()

            try {
                let user
                if (role === 'developer') user = await
Developer.findById(userId).lean()
                else user = await Customer.findById(userId).lean()
                if (!user) reject()
                resolve({ ...user, role })
            } catch (err) {
                reject()
            }
        })
    })
}
```

## utils/file.js

```
const { BadRequest } = require('../errors')
const AWS = require('aws-sdk')
const s3 = new AWS.S3({
    region: 'us-east-2',
    accessKeyId: 'AKIAURGBL3JKZVVQQU7T',
    secretAccessKey: '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'
```

```
})
AWS.config.setPromisesDependency(require('bluebird'))

const parseBase64 = (file) => {
    return {
        data: new Buffer.from(file.slice(file.indexOf(';') + 8), 'base64'),
        type: file.split(';')[0]
    }
}

exports.uploadAvatar = async (file, userId) => {
    if (!file.includes('image')) throw new BadRequest('Image only
allowed!')
    const { data, type } = parseBase64(file)

    const params = {
        Bucket: 'codderasil.avatars',
        Key: `${userId}`,
        Body: data,
        ACL: 'public-read',
        ContentEncoding: 'base64',
        CacheControl: 'no-cache',
        ContentType: type
    }

    const { Location } = await s3.upload(params).promise()
    return Location
}

exports.uploadAttachments = async (files, date, userId) => {
    for (const i in files) {
        for (const j in files) {
            if (files[i].name === files[j].name && i !== j) throw new
BadRequest('Filenames should be unique!')
        }
    }

    const promises = files.map(file => new Promise(async (res, rej) => {
        const { data, type } = parseBase64(file.data)

        const params = {
            Bucket: 'codderasil.attachments',
            Key: `${file.name}.${userId}.${date.getTime()}`,
            Body: data,
            ContentEncoding: 'base64',
            ContentType: type
        }

        try {
            await s3.upload(params).promise()
            res(file.name)
        } catch (err) {
            rej('Failed to upload attachments')
        }
    }))
    return Promise.all(promises)
}

exports.getAttachments = async (keys, date, userId) => {
    const promises = keys.map(key => new Promise(async (res, rej) => {
        const params = {
            Bucket: 'codderasil.attachments',
            Key: `${key}.${userId}.${date.getTime()}`
        }
```

```
        try {
            const data = await s3.getObject(params).promise()
            res({
                name: key,
                data:
`${data.ContentType}; base64,${data.Body.toString('base64')}`
            })
        } catch (err) {
            rej('Failed to load attachments')
        }
    }))
    return Promise.all(promises)
}
```

## utils/rank.js

```
const ranks = [
    { rank: 'F', minimalRate: 4, monthsToUpgrade: 1, downgradeValue: 3 },
    { rank: 'E', minimalRate: 7, monthsToUpgrade: 2, downgradeValue: 5 },
    { rank: 'D', minimalRate: 10, monthsToUpgrade: 2, downgradeValue: 6 },
    { rank: 'C', minimalRate: 15, monthsToUpgrade: 5, downgradeValue: 7 },
    { rank: 'B', minimalRate: 24, monthsToUpgrade: 6, downgradeValue: 8 },
    { rank: 'A', minimalRate: 37, monthsToUpgrade: 8, downgradeValue: 9 },
    { rank: 'S', minimalRate: 50, downgradeValue: 10 }
]

const getAvailableRanks = (rank) => {
    return ranks.slice(0, ranks.indexOf(ranks.find(x => x.rank === rank)) +
1).map(x => x.rank)
}

const getUpgradePercentage = (rank, timeranges) => {
    const summary = timeranges.reduce((prev, current) => prev + current, 0)
/ (1000 * 60 * 60 * 24 * 30)
    const monthsToUpgrade = ranks.find(x => x.rank ===
rank).monthsToUpgrade
    return summary / monthsToUpgrade
}

const getDowngradePercentage = (rank, timeranges) => {
    const summary = timeranges.reduce((prev, current) => prev + current, 0)
/ (1000 * 60 * 60 * 24 * 30)
    const downgradeValue = ranks.find(x => x.rank === rank).downgradeValue
    return (summary * 0.7 + timeranges.length * 0.3) / downgradeValue
}

const getNextRank = (rank) => {
    if (rank === 'S') return
    return ranks[ranks.indexOf(ranks.find(x => x.rank === rank)) + 1].rank
}

const getPreviousRank = (rank) => {
    if (rank === 'F') return
    return ranks[ranks.indexOf(ranks.find(x => x.rank === rank)) - 1].rank
}

module.exports = {
    ranks,
    getAvailableRanks,
    getUpgradePercentage,
    getDowngradePercentage,
    getNextRank,
```

```
    getPreviousRank
}
```

## utils/project.js

```
const { getDowngradePercentage, getUpgradePercentage, getNextRank,
getPreviousRank } = require('./rank')
const { deleteChatByParticipants } = require('./chat')
const { BadRequest } = require('../errors')
const moment = require('moment')

const generateHistory = (developer) => {
    return [
        ...developer.failedProjects.map(x => {
            return {
                applied: x.applied,
                closed: x.closed,
                id: x.id,
                status: 'failed'
            }
        }),
        ...developer.succeedProjects.map(x => {
            return {
                applied: x.applied,
                closed: x.closed,
                id: x.id,
                status: 'closed'
            }
        })
    ]
}

exports.getNewDeadline = (created, deadline) => {
    const newDeadline = new Date()
    newDeadline.setTime(newDeadline.getTime() + (new Date(deadline) - new
Date(created)))
    return newDeadline
}

// used both by developer controller (to cancel the project) and customer
controller (to decline the project)
exports.declineProject = async (req, res, developer, project) => {
    developer.failedProjects.push({
        applied: project.applied,
        closed: new Date(),
        id: project._id
    })

    const timeranges = developer.failedProjects.map(project => new
Date(project.closed) - new Date(project.applied))
    if (getDowngradePercentage(developer.rank, timeranges) >= 1) {
        const rank = getPreviousRank(developer.rank)
        if (!rank) developer.banned = true
        else {
            developer.rank = rank
            developer.history                                          =
developer.history.concat(generateHistory(developer))
            developer.succeedProjects = []
            developer.failedProjects = []
        }
    }

    await developer.save()
```

```
    await deleteChatByParticipants([project.owner, project.developer])

    project.status = 'pending'
    project.developer = undefined
    project.deadline = this.getNewDeadline(project.created,
project.deadline)
    if (project.milestones) {
        for (const milestoneIndx in project.milestones) {
            project.milestones[milestoneIndx].deadline =
this.getNewDeadline(project.created,
project.milestones[milestoneIndx].deadline)
        }
    }
    await project.save()
    res.json(project)
}

exports.closeProject = async (req, res, developer, project) => {
    developer.succeedProjects.push({
        applied: project.applied,
        closed: new Date(),
        id: project._id
    })

    const timeranges = developer.succeedProjects.map(project => new
Date(project.closed) - new Date(project.applied))
    if (getUpgradePercentage(developer.rank, timeranges) >= 1) {
        const rank = getNextRank(developer.rank)
        if (rank) {
            developer.rank = rank
            developer.history = developer.history.concat(developer)
            developer.succeedProjects = []
            developer.failedProjects = []
        }
    }

    await developer.save()

    await deleteChatByParticipants([project.owner, project.developer])

    project.status = 'closed'
    project.closed = new Date()
    project.developer = undefined
    await project.save()
    res.json(project)
}

exports.returnProject = async (req, res, developer, project) => {
    if (project.isOverdue) throw new BadRequest('You should firstly
continue deadlines of the project')
    project.status = 'returned'
    await project.save()
    // TODO: notify developer
    res.json(project)
}

exports.approveMilestone = async (req, res, developer, project) => {
    const currentMilestone = project.milestones.find(x => !x.finished)
    const currentMilestoneIndx =
project.milestones.indexOf(currentMilestone)
    project.milestones[currentMilestoneIndx].finished = true
    if (currentMilestoneIndx === project.milestones.length - 1) {
        await project.save()
```

```
        return this.closeProject(req, res, developer, project)
    } else {
        project.status = 'active'
        if (project.isOverdue) throw new BadRequest('You should firstly
continue deadlines of the project')
        await project.save()
    }
    res.json(project)
}

const countBusinessDays = (start, end) => {
    const isWeekend = (date) => date.day() === 0 || date.day() === 6
    start = moment(start)
    end = moment(end)
    const daysInFirstWeek = isWeekend(start) ? 0 : 5 - start.day()
    const daysInLastWeek = end.week() === start.week() ? 0 :
(isWeekend(end) ? 5 : end.day() - 1)

    const weeksDiff = end.week() - start.week()
    let restDays = weeksDiff > 0 ? (weeksDiff - 1) * 5 : 0
    return daysInFirstWeek + restDays + daysInLastWeek
}

exports.getMinimalBudget = (deadline, minimalRate) => {
    const weekDays = countBusinessDays(new Date(), new Date(deadline))
    const hoursToWork = weekDays * 8
    const minimalBudget = parseInt(minimalRate * hoursToWork)
    return minimalBudget > 0 ? minimalBudget : null
}
```

**Додаток 3**
**Копія презентації**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

# ВЕБ-СЕРВІС ДЛЯ ПОШУКУ ДИСТАНЦІЙНОЇ РОБОТИ НА ОСНОВІ СИСТЕМИ РАНГІВ

## Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Яків ЮСИН

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Андріан ПОЙДА

2019

**ЗМІСТ**

## 1. ОБ’ЄКТ ВИПРОБУВАНЬ

Веб-сервіс для пошуку дистанційної роботи на основі системи рангів, який являє собою веб-сайт, клієнтська частина якого розроблена на React, а серверна – на Node.js.

## 2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність елементів сторінок веб-ресурсу;
2) забезпечення належного рівня безпеки даних;
3) зручність роботи з веб-сайтом;
4) відповідність додатка вимогам Технічного завдання.

## 3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1) функціональне тестування, зокрема на рівні Critical path test (базове тестування);
2) тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);
3) мануальне тестування інтерфейсу.

# 4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування серверної частини виконується за допомогою використання Mocha, Chai та Supertest. Для тестування клієнтської частини було виконано мануальний підхід.

Працездатність веб-додатка перевіряється шляхом:

1) динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
2) динамічного ручного тестування на відповідність функціональним вимогам;
3) статичного тестування коду;
4) тестування веб-ресурсу в різних браузерах;
5) тестування при максимальному навантаженні;
6) тестування стабільності роботи при різних умовах;
7) тестування зручності використання;
8) тестування інтерфейсу.

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

# ВЕБ-СЕРВІС ДЛЯ ПОШУКУ ДИСТАНЦІЙНОЇ РОБОТИ НА ОСНОВІ СИСТЕМИ РАНГІВ

**Керівництво користувача**

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Яків ЮСИН

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Андріан ПОЙДА

2020

**ЗМІСТ**

## 1. Опис вмісту веб-сторінок для незареєстрованого користувача

На головній сторінці незареєстрованого користувача відображається загальна інформація про платформу (рис. 1), кнопка «Sign in» для авторизації, та кнопка «Sign up» (або «Get Started») для реєстрації (рис. 2).

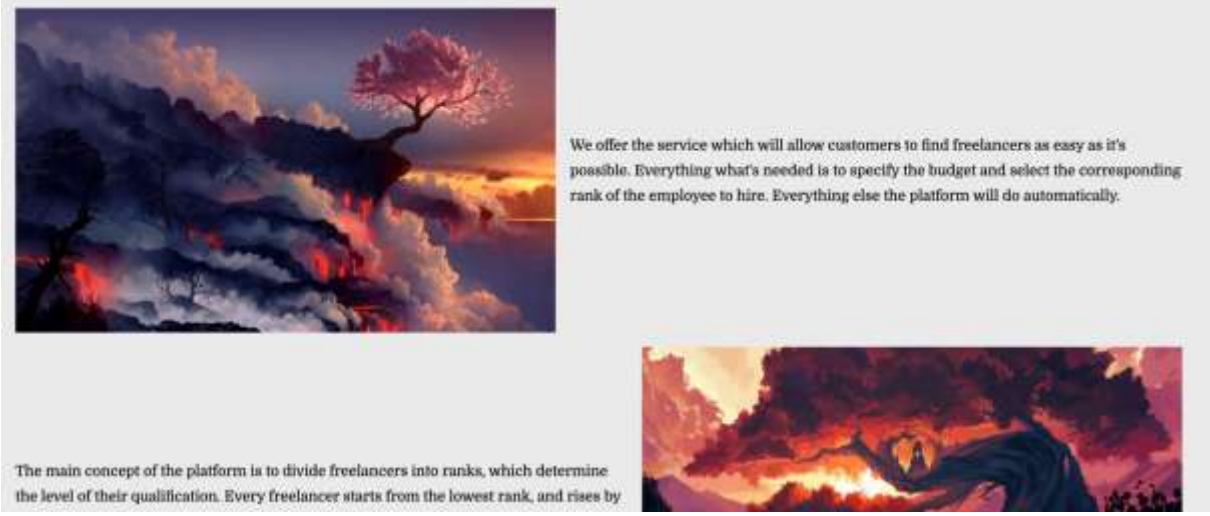



Рис. 1. Загальний опис платформи на головній сторінці

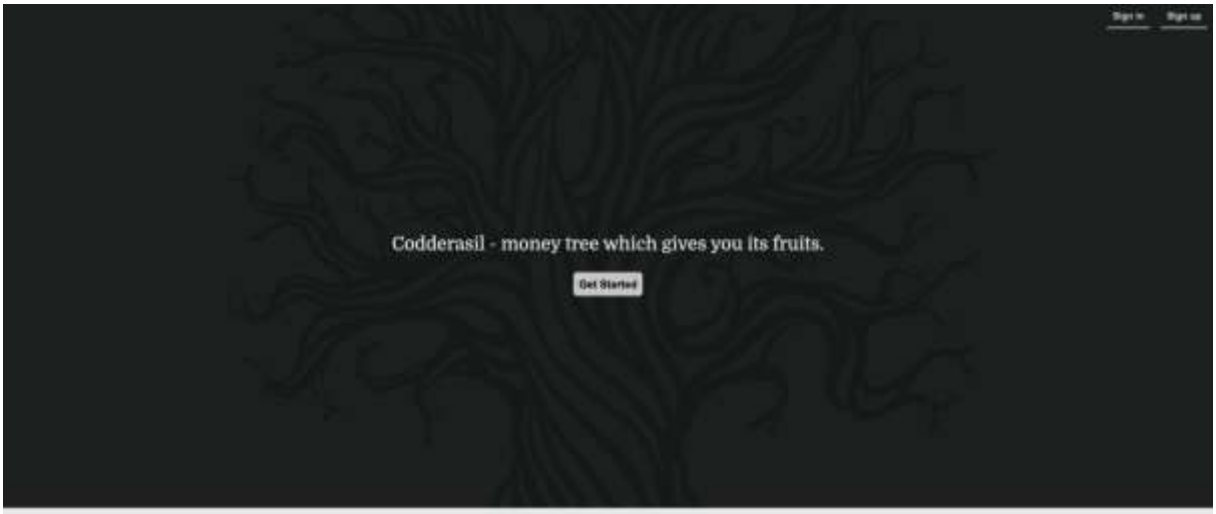



Рис. 2. Заголовок головної сторінки з кнопками для авторизації та реєстрації

На сторінці реєстрації користувачу пропонується обрати роль (рис. 3) після чого ввести персональні дані (рис. 4).


Рис. 3. Вибір ролі користувача

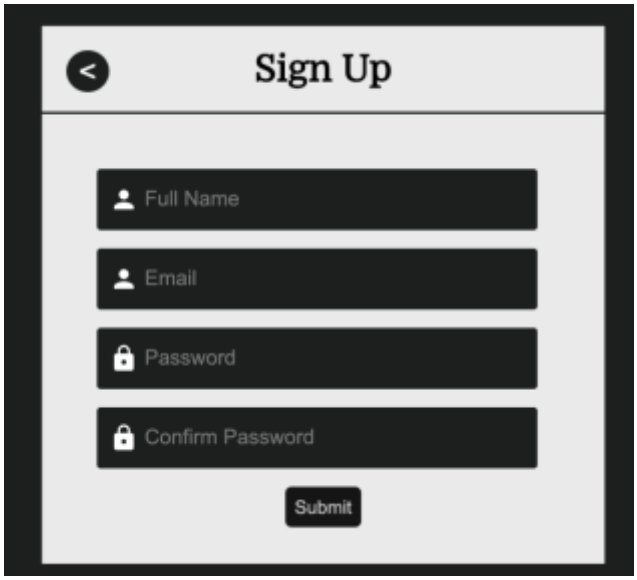



Рис. 4. Форма реєстрації

На сторінці авторизації відображається форма для введення електронної пошти та паролю (рис. 5).

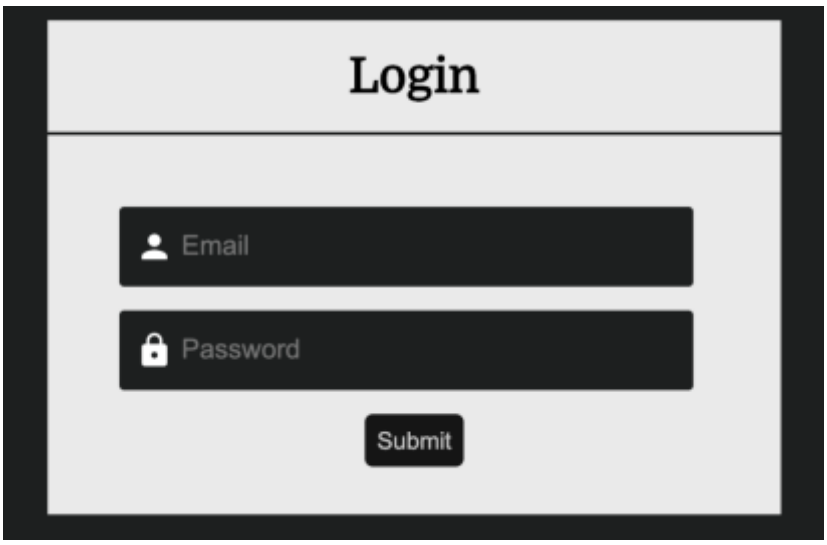



Рис. 5. Форма авторизації

## 2. Опис вмісту веб-сторінок для замовника

На кожній сторінці замовника відображається меню з аватаром, кнопкою для публікації нового замовлення, кнопкою для виходу з облікового запису та панеллю для навігації по сторінкам (рис. 6).

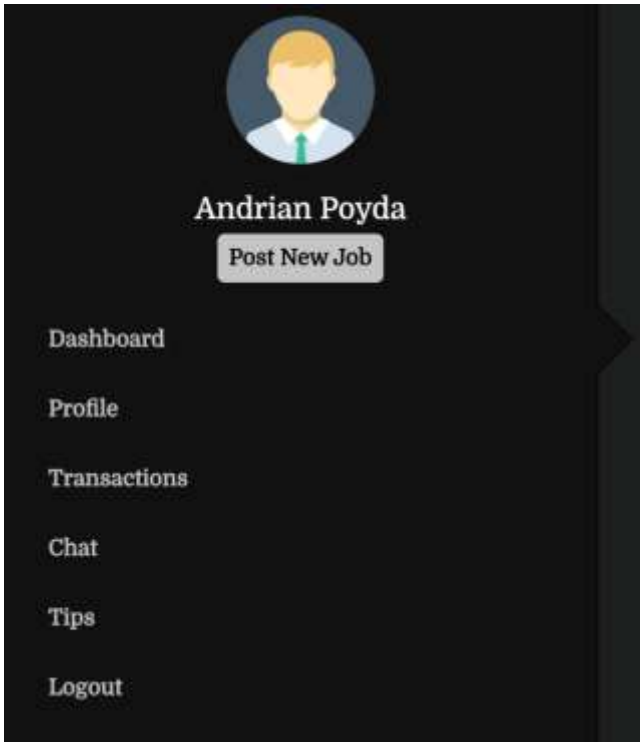



Рис. 6. Меню користувача

На головній сторінці відображається список опублікованих замовлень (рис. 7) із наступною інформацією:

- назва проєкту;
- кінцевий строк;
- бюджет, виділений на розробку проєкту;
- ранг працівника, який розроблятиме продукт;
- статус замовлення.

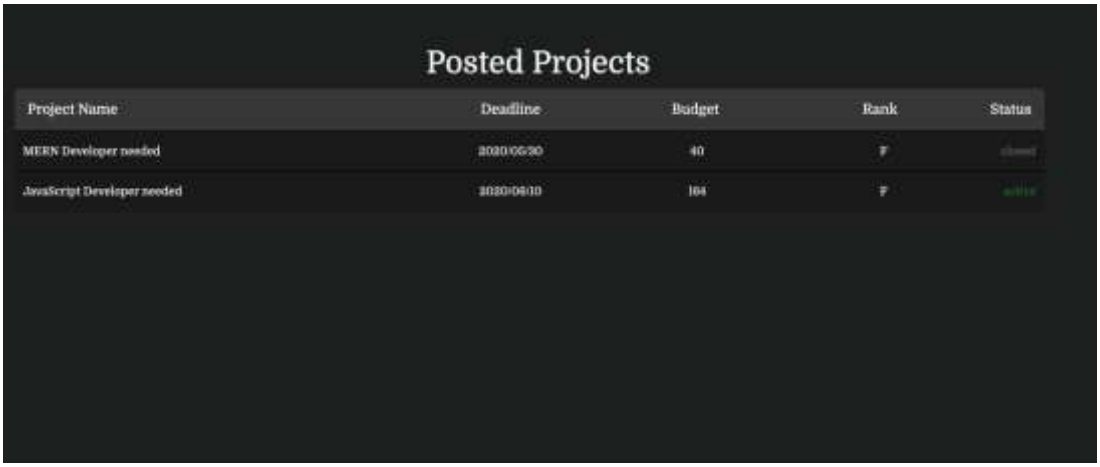



Рис. 7. Список опублікованих замовлень

При натисненні на будь-який проєкт зі списку, випливає вікно з деталями проєкту (рис. 8).


Рис. 8. Деталі проєкту

В даному вікні в користувача є можливість редагувати кінцеві строки проєкту, видаляти замовлення (у випадку, якщо жоден працівник не працював над ним), схвалювати, відправляти на доопрацювання чи відхиляти роботу працівника.

Для публікації нового замовлення, користувачу необхідно натиснути на кнопку «Post New Job», що розташована в меню, після чого з’явиться вікно для заповнення даних (рис. 9).

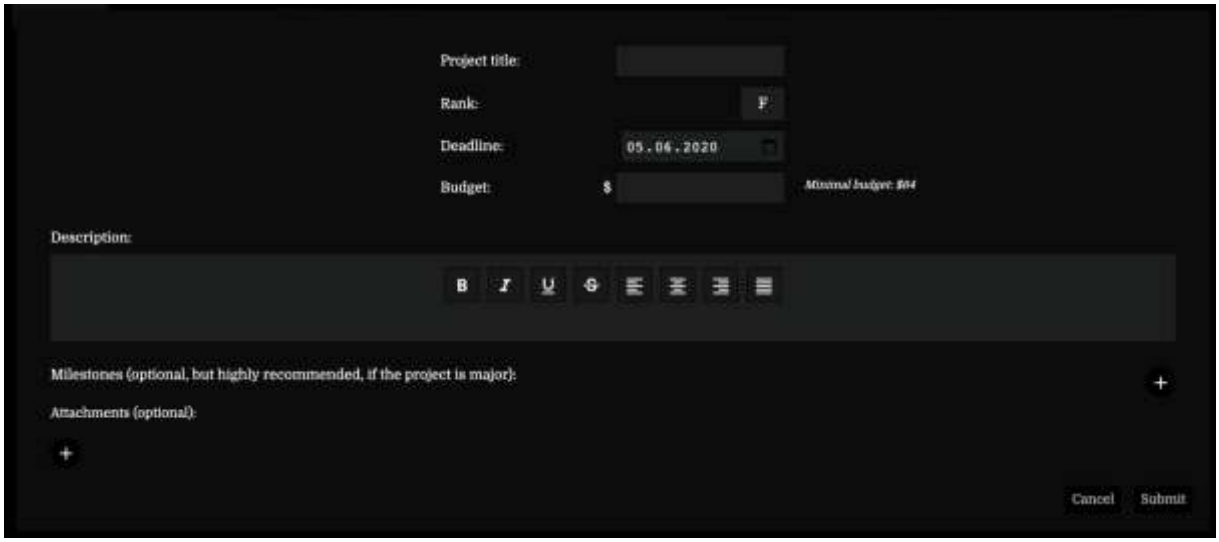



Рис. 9. Форма для публікації нового проєкту

Окрім заповнення загальної інформації про проєкт, є можливість його розділення на кілька етапів. Для цього існує розділ «Milestones», в якому можна добавити необмежену кількість блоків з інформацією про кожен етап (рис. 10).


Рис. 10. Етапи розроблення проєкту

Після заповнення всіх полів та здійснення оплати на суму, вказану в полі «Budget», вікно закриється, а замовлення опублікується на дошці об’явлень відповідного рангу. Також воно з’явиться в списку опублікованих замовлень, а інформація про здійснену оплату відображатиметься в списку транзакцій (рис. 12).

На сторінці профілю відображається інформація про користувача та спосіб оплати (див. рис. 11), на якій є можливість оновити спосіб оплати, ім’я, електронну пошту, чи пароль користувача.

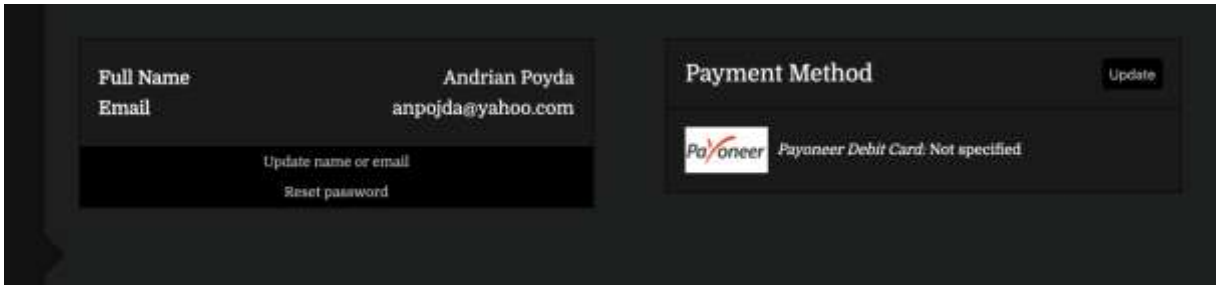



Рис. 11. Сторінка профілю.

На сторінці транзакцій є можливість переглянути список проведених транзакцій з інформацією про замовлення (рис. 12).

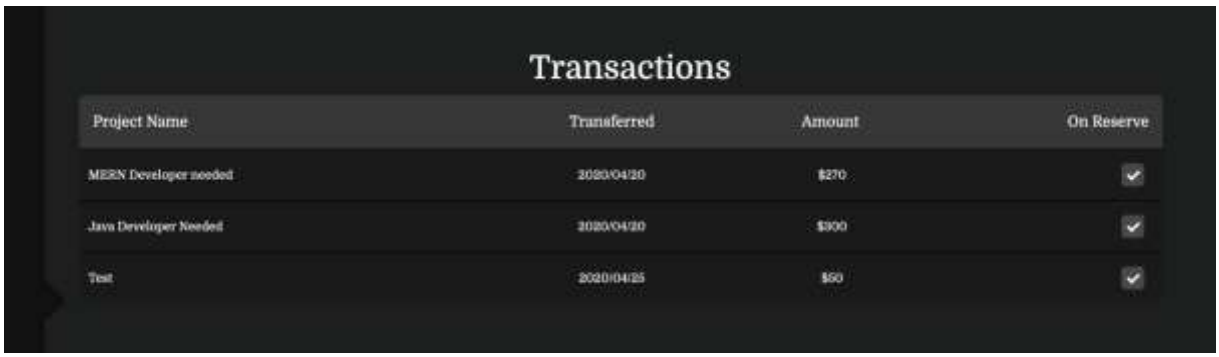



Рис. 12. Сторінка транзакцій

На сторінці чату відображається чат для підтримки зв’язку з працівниками (рис. 25).

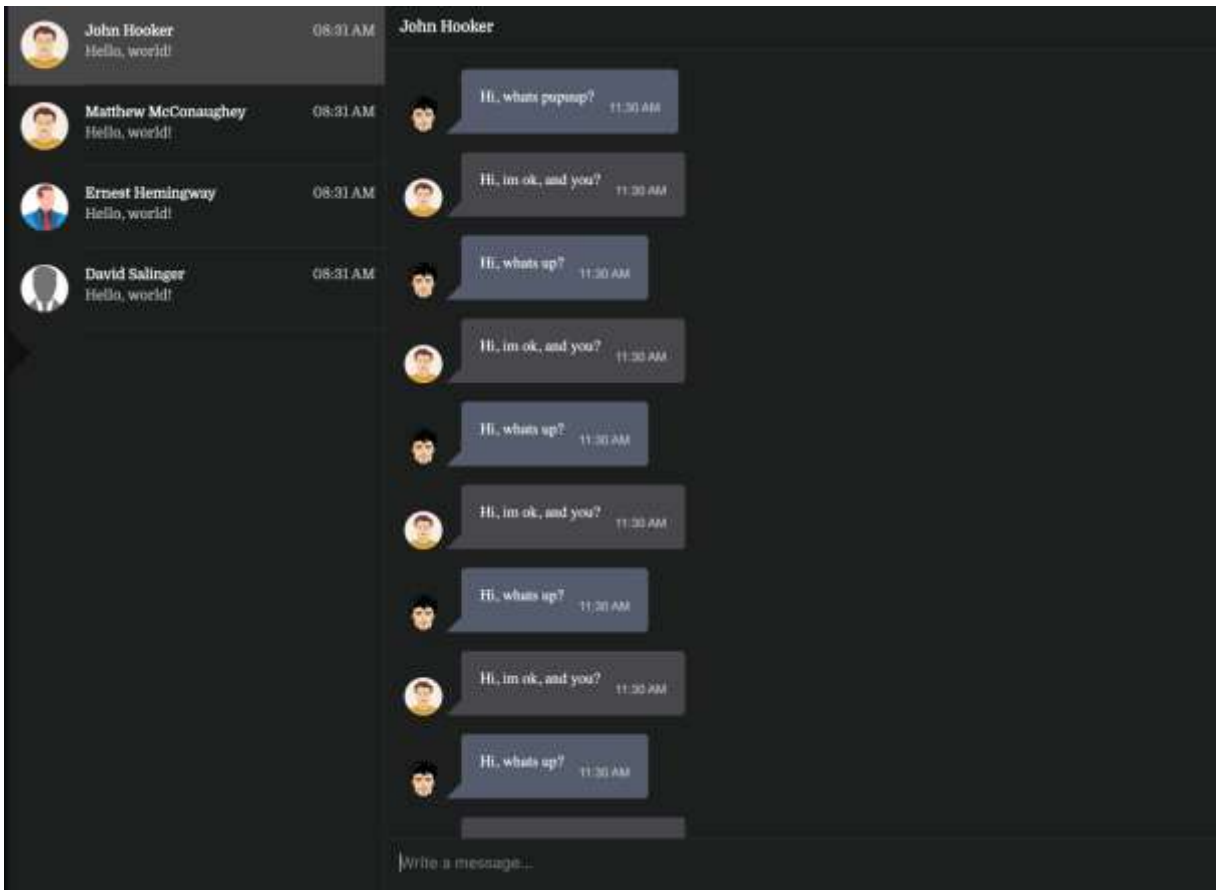



Рис. 13. Чат

На сторінці Tips розташовані рекомендації для користувача щодо користування платформою.

## 2. Опис вмісту веб-сторінок для працівника

На кожній сторінці відображається меню з аватаром, рангом користувача, кнопкою для виходу з облікового запису та панеллю для навігації по сторінкам (рис. 14).

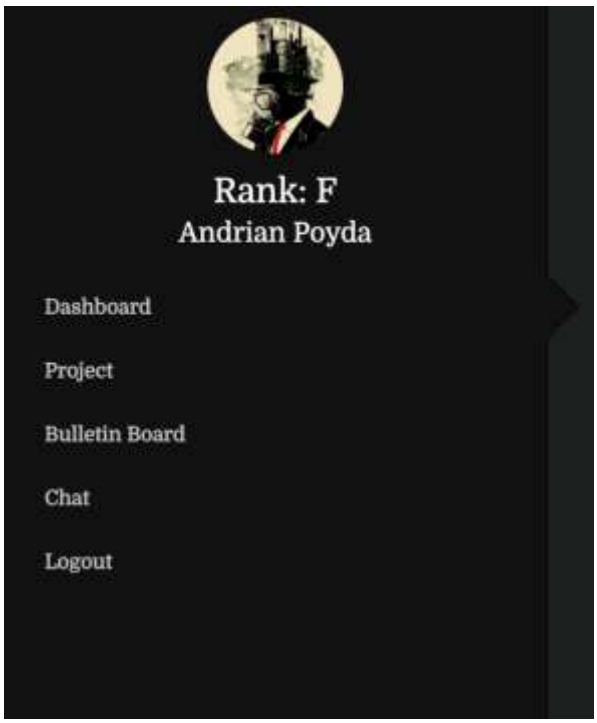



Рис. 14. Меню користувача

На головній сторінці (рис. 15) відображається прогрес користувача:

- існуючі ранги та пройдені серед них працівником;
- відсоток успішно завершених проєктів до загальної кількості, за якою слідує підвищення рангу;
- відсоток провалених проєктів до загальної кількості, за якою слідує пониження рангу / видалення облікового запису із системи.

Також на сторінці є кнопки, при натисненні на які появляється вікно (рис. 16) з наступною інформацією:

- статистика активності працівника та його доходів;
- статистика успішності проєктів;
- профіль користувача з можливістю його редагування;
- спосіб оплати;
- рекомендації щодо користування платформою.

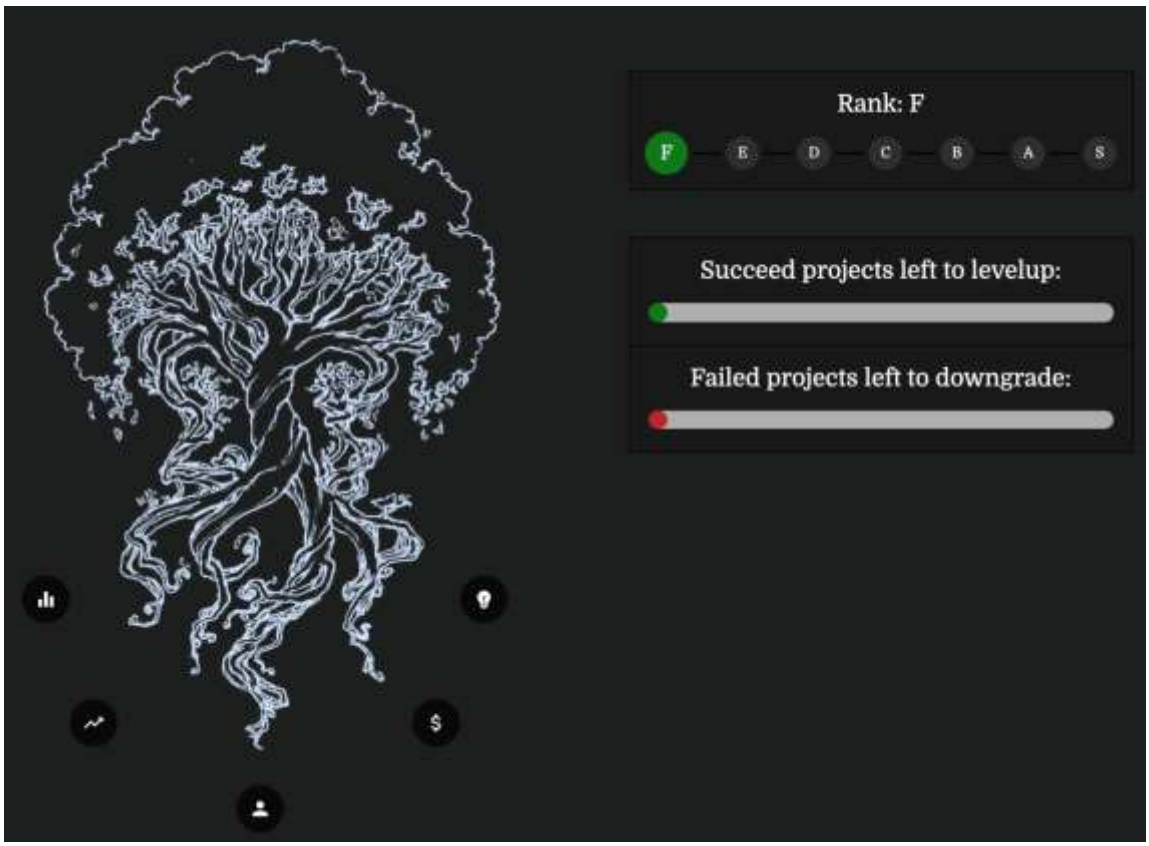



Рис. 15. Головна сторінка працівника

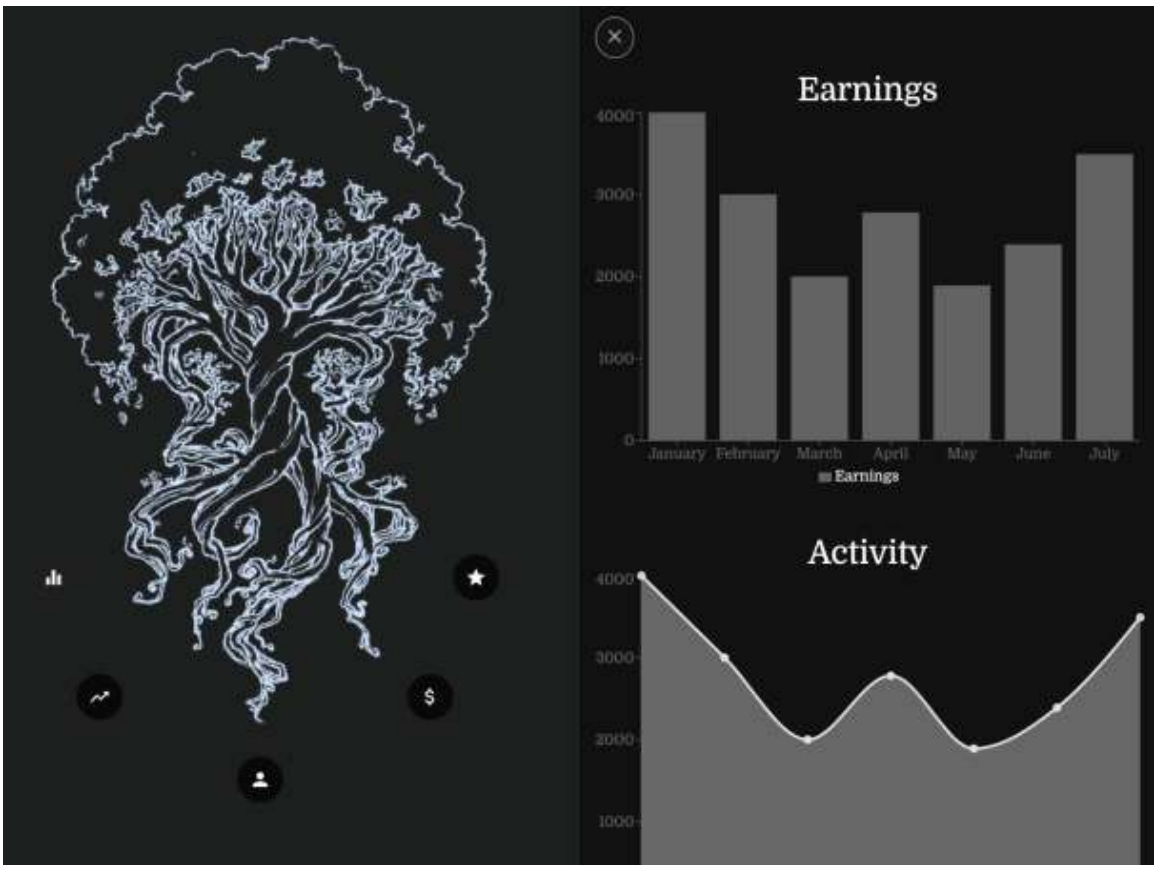



Рис. 16. Вікно з відображенням інформації на головній сторінці

На сторінці проєкту відображається детальна інформація про проєкт, на який подав працівник, та можливість його відхилення чи подачі результатів роботи (рис. 17).

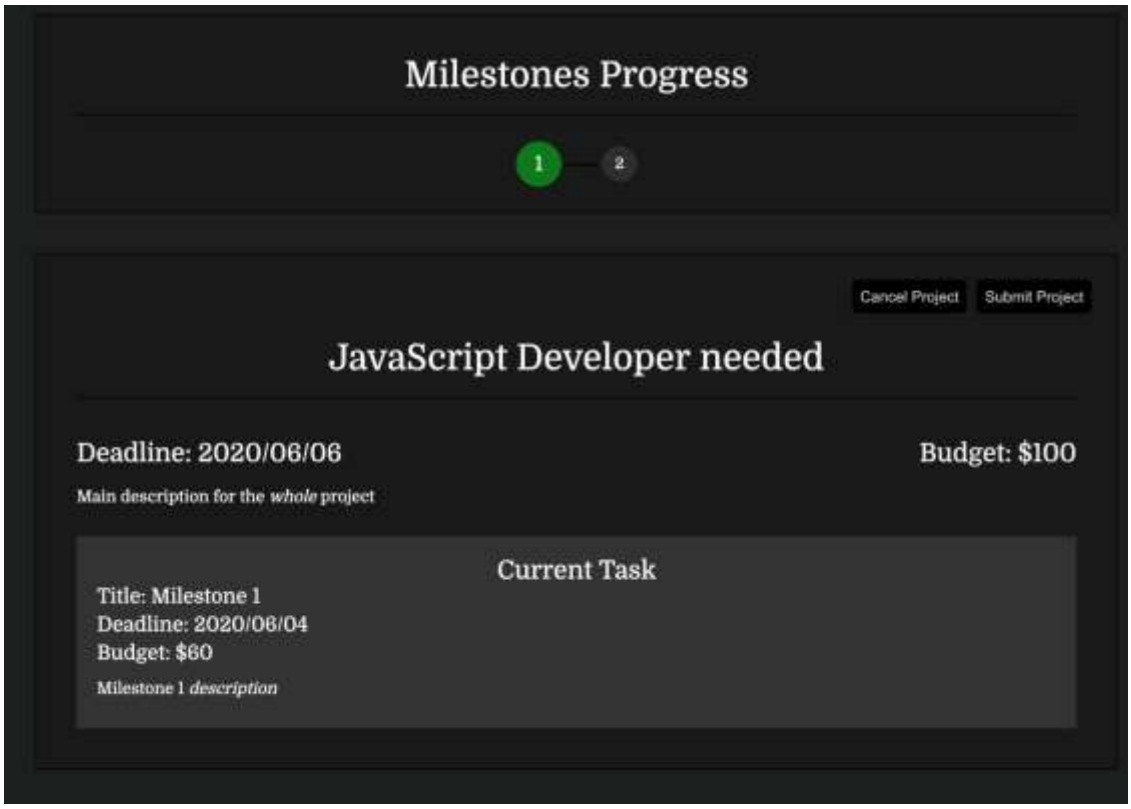



Рис. 17. Сторінка проєкту, над яким працює фрілансер

На сторінці дошки об’явлень відображаються вільні замовлення відповідного рангу (рис. 18).

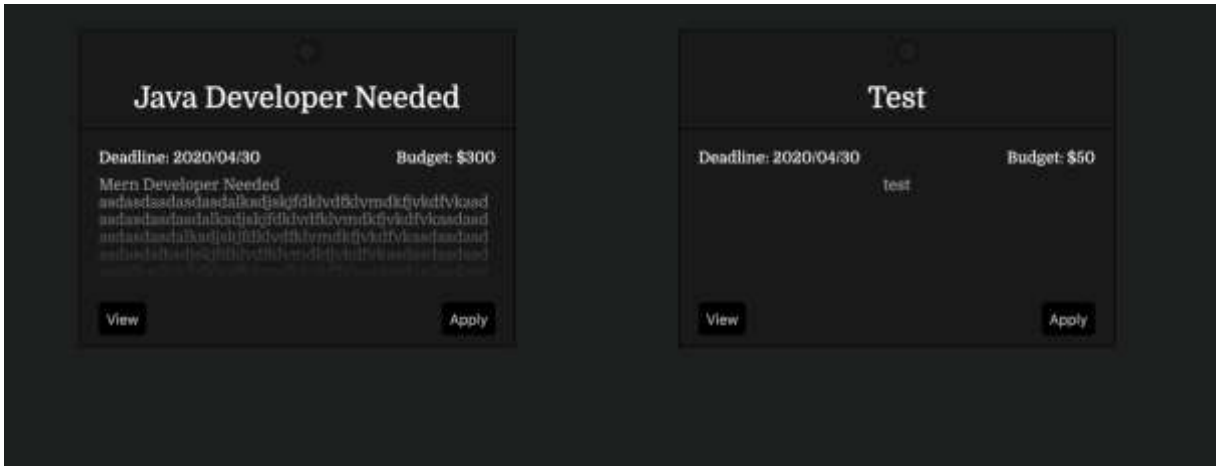



Рис. 18. Список опублікованих замовлень

При натисненні на кнопку View відкривається вікно з детальною інформацією про проєкт (рис. 19).


Рис. 19. Детальна інформація про проєкт на дошці об’явлень

В правому верхньому кутку сторінки є кнопка, при натисненні на яку відкриється вікно для налаштування фільтрації проєктів (рис. 20).


Рис. 20. Налаштування фільтрації проєктів

На сторінці чату відображається чат для підтримки зв’язку із замовниками (рис. 21).

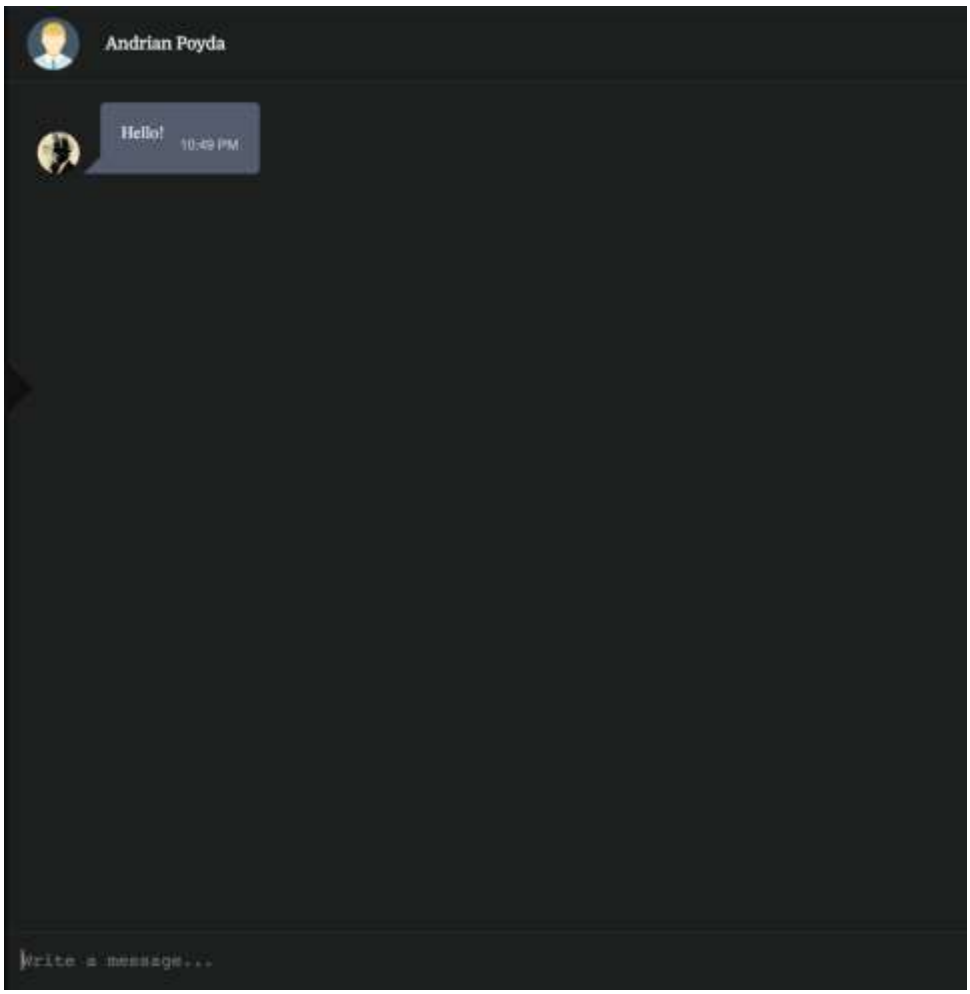



Рис. 21. Чат